



# User's Guide TDCompress<sup>TM</sup>

---

Copyright © 2001 by bTrade, Inc.

All Rights Reserved.

Printed in the USA

Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in any data base or retrieval system, without the prior written permission of bTrade, Inc.

---

For additional information, contact:

bTrade, Inc.  
2324 Gateway Drive  
Irving, Texas 75063  
[800-425-0444](tel:800-425-0444)  
or by e-mail:  
[support@btrade.com](mailto:support@btrade.com)

---

# Table of Contents

<b>CONTACTING US .....</b>	<b>1-1</b>
Technical Support .....	1-1
<i>Technical Support Call Checklist</i> .....	1-2
 <b>INTRODUCTION.....</b>	 <b>1-5</b>
Why You Should Use TDCompress.....	1-5
TDCompress Features and Options.....	1-5
TDCompress Extended Security Option.....	1-5
Master or Distribution License.....	1-6
TDCompress Features.....	1-6
Operating Environments.....	1-7
 <b>CHAPTER 1:    BEFORE YOU BEGIN .....</b>	 <b>1-8</b>
Considerations.....	1-8
ASCII-to-EBCDIC Translation .....	1-8
Handling Carriage Return/Line Feed Pairs .....	1-8
Transferring Files Across Platforms .....	1-9
File Formats and Names .....	1-9
Upgrading to New Versions of TDCompress.....	1-9
Upgrading From Previous Versions .....	1-10
Comm-Press 3.4.x and 4.4.x Migration Guide .....	1-10
Secret Key Values.....	1-10
Secret Keys with Initialization Vectors.....	1-11
 <b>CHAPTER 2:    SECURING DATA USING TDCOMPRESS.....</b>	 <b>2-12</b>
Execution Options for TDCompress Security Processing.....	2-12
Extended Security Option Overview .....	2-14
Overview of Secret Key Encryption.....	2-15
An Overview of Public/Private Key Encryption.....	2-15
Using Public/Private Key Technology to Encrypt Data.....	2-16
Digital Signatures and Authentication.....	2-17
Filtering .....	2-17
Encrypting Data with TDCompress.....	2-18
Specifying the Secret Key and Initialization Vector.....	2-18
Supplying the Key and IV on the Command Line .....	2-18
Supplying the Key and IV in a File .....	2-19
 <b>CHAPTER 3:    COMPRESS AND DECOMP OPTIONS.....</b>	 <b>3-20</b>
Compress Options.....	3-21
Common .....	3-21
TDCompress Extended Security Options.....	3-22
Decompress Options .....	3-22
Common .....	3-22
TDCompress Extended Security Option.....	3-23
Compress/Decompress Option Descriptions.....	3-24
 <b>CHAPTER 4:    KEY GENERATION AND TDMANAGER RUN-TIME FILES (TDCOMPRESS EXTENDED SECURITY OPTION).....</b>	 <b>4-32</b>
RSA Key Generation.....	4-32
TDManager Run-time Files .....	4-33

---

The Certificate Run-time File .....	4-33
The Private Key Run-time File .....	4-33
The Symmetric Key Run-time File .....	4-34
The Lookup Table Run-time File .....	4-34
The Participant Table Run-time File .....	4-35
<b>CHAPTER 5: SECFILE KEYWORDS BY PLATFORM.....</b>	<b>5-36</b>
MVS .....	5-36
SECFILE Keywords for Data with User-Defined Headers .....	5-36
SECFILE Keywords for Data with No Headers .....	5-38
MVS Example for Securing Non-EDI Data .....	5-38
PC .....	5-39
SECFILE Keywords for Data with User-Defined Headers .....	5-39
SECFILE Keywords for Data with No Headers .....	5-40
PC Example for Securing Non-EDI Data .....	5-41
<b>CHAPTER 6: MVS PLATFORM.....</b>	<b>6-42</b>
MVS Installation .....	6-42
Using a diskette or CD-ROM .....	6-42
Using a 3480 tape cartridge.....	6-43
Binding the DB2 Plan (Extended Security Option) .....	6-43
Installing the TDManager Run-time Files .....	6-44
Using IMPORT on MVS .....	6-44
Mainframe Operation .....	6-44
Compressing on the Mainframe .....	6-45
Decompressing on the Mainframe .....	6-46
Using ARCHIVE with DECOMP.....	6-48
Using Dynamic Allocation with DECOMP.....	6-48
Modifying the Default Translation Tables at Run Time .....	6-50
Using GENKEYS on MVS .....	6-53
Securing Formatted EDI Data .....	6-53
MVS Example for Securing Formatted EDI Data .....	6-53
Unsecuring Formatted EDI Data.....	6-54
MVS Example for Unsecuring Formatted EDI Data.....	6-54
Securing Non-EDI Data .....	6-54
Unsecuring Non-EDI Data .....	6-55
MVS Example for Unsecuring Non-EDI Data .....	6-55
Compressing Mainframe Files .....	6-55
Modifying the Default Translation Tables .....	6-56
<b>CHAPTER 7: VMS PLATFORM.....</b>	<b>7-58</b>
Installation.....	7-58
TK7 .....	7-58
Disk .....	7-58
VMS Operation .....	7-59
Compressing on VMS .....	7-60
Decompressing on VMS.....	7-61
VMS Examples .....	7-61
<b>CHAPTER 8: WINDOWS 95/98/NT PLATFORMS .....</b>	<b>8-63</b>
Installation.....	8-63
Diskette or CD.....	8-63
Installing the TDManager Run-time Files .....	8-63
Using IMPORT on Windows 95/98/NT .....	8-63

---

PC Operation .....	8-64
Compressing on the PC (DOS, OS/2 and Windows).....	8-65
Decompressing on the PC (DOS, OS/2 and Windows).....	8-66
PC Compression/Decompression Examples .....	8-67
Compressing PC/Workstation Files.....	8-68
Using GENKEYS On Windows 95/98/NT .....	8-68
Input to GENKEYS .....	8-69
Output from GENKEYS .....	8-70
Securing Formatted EDI Data .....	8-70
PC Example for Securing Formatted EDI Data .....	8-70
Unsecuring Formatted EDI Data.....	8-71
PC Example using ARCHIVE with Secure Formatted EDI Data .....	8-71
PC Example for Unsecuring Formatted EDI Data .....	8-71
Securing Non-EDI Data .....	8-72
Unsecuring Non-EDI Data .....	8-72
PC Example for Unsecuring Non-EDI Data .....	8-72
Compressing EDI-Formatted Data .....	8-73
 <b>CHAPTER 9:    UNIX/AIX PLATFORM.....</b>	<b>9-74</b>
Installation.....	9-74
Diskette/Tape.....	9-74
Installing the TDManager Run-time Files .....	9-75
Using IMPORT on UNIX .....	9-75
UNIX Operation.....	9-76
Compressing on UNIX/AIX .....	9-77
Decompressing on UNIX/AIX.....	9-77
UNIX/AIX Examples.....	9-78
Using GENKEYS on UNIX .....	9-79
Input to GENKEYS .....	9-81
Output from GENKEYS .....	9-81
 <b>CHAPTER 10:    OS/400 PLATFORM.....</b>	<b>10-82</b>
Installation.....	10-82
Diskette.....	10-82
Tape.....	10-82
Installing the TDManager Run-time Files .....	10-83
Using IMPORT on OS/400 .....	10-83
AS/400 Operation .....	10-83
Compressing on the AS/400 .....	10-84
Decompressing on the AS/400 .....	10-85
Using GENKEYS on OS/400 .....	10-86
Modifying the Default Translation Tables .....	10-88
 <b>APPENDIX A .....</b>	<b>10-89</b>
Using API (Extended Security Option).....	10-89
Using the Command-Line API (COMPRESS and DECOMP) .....	10-89
Using the Interactive API (compprog and dcmpprog) .....	10-90
The Interactive API Conversation.....	10-91
Ending the Conversation .....	10-92
 <b>APPENDIX B .....</b>	<b>10-93</b>
Technical Notes.....	10-93

---

<b>APPENDIX C .....</b>	<b>10-95</b>
Error Messages and Codes .....	10-95
COMPRESS and DECOMP Error Messages and Codes.....	10-95
Secondary Messages/Return Codes .....	10-108
AUTACK Error Codes .....	10-109
Additional API Error Codes.....	10-112
BSAFE Return Codes .....	10-112
 <b>APPENDIX D .....</b>	 <b>10-117</b>
Support CCA .....	10-117
Overview.....	10-117
Prerequisites.....	10-117
Operation.....	10-117
 <b>GLOSSARY OF TERMS .....</b>	 <b>10-118</b>
 <b>INDEX .....</b>	 <b>122</b>

# Contacting Us

## Technical Support

Customer Services is available 24 hours a day, seven days a week, based on the following criteria:

- **Prime Support Hours** — 7 a.m. – 6 p.m. (CST) Monday through Friday.
  - Customer Service Representatives are available to answer your calls and assist you with a comprehensive range of services during these hours.
- **After Hours Support** — Available any time outside the prime support hours, including nights, weekends, and holidays. Representatives are on-call to respond to the Severity 1 issues that cannot wait for the next business day.
  - **800-425-0444** for customers in North America.
  - **972-580-2900** for customers outside of North America.

## Technical Support Call Checklist

Before contacting Technical Support please fill out the information below. Send this information and the Pre-Installation Worksheet to Technical Support concerning issues. If an error occurs and there is a need to contact Technical Support, please have the following information and the Pre-installation worksheet available. For more in-depth issues, send the information collected to [support@btrade.com](mailto:support@btrade.com).

### Contact Information:

Company Name: \_\_\_\_\_

Company Address1: \_\_\_\_\_

Company Address2: \_\_\_\_\_

Contact Name: \_\_\_\_\_

Contact Primary Phone: \_\_\_\_\_

Contact Secondary Phone: \_\_\_\_\_

Contact e-mail address: \_\_\_\_\_

### Product:

TDAccess v1.x ☐

TDAccess v2.x ☐

TDManager ☐

TDServer ☐

### Platform:

*Intel-based*

Windows2000 ☐

WindowsXP ☐

WindowsNT ☐

Windows98 ☐

*UNIX*

HP-UX ☐

AIX ☐

SUN ☐

*Mainframe*

MVS – OS/390 ☐

AS400 ☐

## License Type:

- Trial ☐
- License ☐
- Sponsor ☐

Hub Trading Partner: \_\_\_\_\_

Symptoms (Provide a detailed **description**):

---

---

---

---

Severity (1, 2, 3, or 4): ☐ Sev 1 ☐ Sev 2 ☐ Sev 3 ☐ Sev 4  
(1 = Critical, 2 = Production affected, 3 = Configuration or setup, 4 = General question)

## Configuration files:

- Tdclient.ini ☐
- Tdserver.cfg ☐
- Tdmanager.ini ☐

## Logs gathered:

- eacomm.log ☐
- eaxfer.log ☐
- ea2k.log ☐
- TDServer log file ☐

## Other Logs gathered:

- firewall ☐
- snoop ☐
- intrusion detection ☐
- other ☐

## Diagnostics used:

- ping ☐
- tracert ☐
- generic ftp ☐

## Has the hub been contacted?

Is hub involved? ☐ Yes ☐ No

Hub contact name: \_\_\_\_\_

Hub contact number: \_\_\_\_\_

Hub contact e-mail address: \_\_\_\_\_

## Documentation:

1. What is the product and release version and list of products being used?
2. Has this been working and just failed or is this a new setup?  
Yes or No
3. Type of connectivity i.e., dialup, SSL, ftp
4. Open the tdaccess.ini, which is located in the root of TDClient directory with a text editor. A point to express is to make sure that the GUI interface is closed when performing this change. Once the tdaccess.ini is open locate the [IDENTIFY] section and make the following changes:

```
LOG_INI=-6
LOG_XFER=-6
LOG_FTP=-6
LOG_EASYACC=-6
```

When those changes have been completed, save the file; open the GUI and run a transfer. Once the transfer is complete the following logs will be created in the following directories these are the files that need to be requested from the Trading Partner.

·	ea2k.log	(Located in the tdclient root directory)
·	eaini.log	(Located in the tdclient root directory)
·	eaxfer.log	(Located in the tdclient temp directory)
·	eacomm.log	(Located in the tdclient temp directory)
·	list.fil	(Located in the tdclient temp directory)
·	tmplist.fil	(Located in the tdclient temp directory)
·	decomp.log	(Located in the tdclient temp directory)

Once Technical Support obtains these logs, it will provide all the information required to evaluate the transaction for its failure.

# Introduction

TDCompress is a set of utilities that allows you to compress, encrypt, authenticate and assure data files for cross-platform file transfers over public and private networks. Cross-platform means that data can be compressed and secured on one type of computer, for instance an IBM mainframe running MVS and then be decompressed and unsecured on a different type of computer, such as a PC running Windows 95. TDCompress takes care of the differences in data formatting between different computer systems by executing the utilities independently of the communication network products, hardware, and operating systems in use. Details on all execution parameters as well as coding programs that invoke the application programming interface (API), are given in the appropriate user guide sections.

## Why You Should Use TDCompress

TDCompress can dramatically cut both the expense and time associated with bulk data transfer. Using state-of-the-art data compression techniques, the size of the data files are reduced by as much as 95 percent. Encryption is provided for secure data transmission.

The compressed data can be transmitted and decompressed between different computing platforms, including mainframes, AS/400s, AIX/UNIX-based systems, and PCs.

Since the compression and decompression takes place off-line, TDCompress is not dependent on any particular communications hardware, software, or protocol.

Used in conjunction with the TDManager product, TDCompress provides state-of-the-art public key technology for encrypting and authenticating data. TDCompress is also interoperable with legacy security products such as Dataguard from Sterling Software.

## TDCompress Features and Options

### TDCompress Extended Security Option

TDCompress Extended Security Option utilities provide advanced functions for handling public/private key pairs and certificates from Certificate Authorities (CA). Public/private keys may be created by the Extended Security Option utilities so that only the public component is provided to the CA for certification. The Extended Security Option utilities support EDI and EDIFACT standard formats, as well as custom requirements. The current software version numbers are 4.n.n. The version number is displayed in the compress.log or decomp.log when the utilities are executed. An example is Version 4.4.2 \*i\*.

## Master or Distribution License

TDCompress provides Master and Distribution licenses. The Master license utilities are unrestricted with regard to both **COMPRESS** or **DECOMP**. **DECOMP** processes data that is compressed/encrypted with either a Master or Distribution licensed product. The Distribution license utilities are unrestricted with regard to **COMPRESS**. **DECOMP** is restricted to process only data that is compressed/encrypted with a Master licensed product.

To determine which type of license the product has, refer to the TDCompress `Readme` file.

## TDCompress Features

Some of the features of TDCompress are:

- Multiple files can be compressed into a single file for transmission. Decompression splits the data back into separate files.
- The ability to compress and decompress X12, EDIFACT, UN/EDI, and UCS EDI data. The EDI envelope records are left in the clear to allow routing through a value-added network (VAN).
- State-of-the-art RSA public key technology for encryption and digital signatures. **DES**, Triple **DES**, and **RC2** algorithms are used for bulk data encryption.
- Built-in options that allow cross-platform data interoperability, for example ASCII/EBCDIC translation and handling of record delimiters.
- An application programming interface (API) that helps the user to precisely control compression/decompression.
- Support for MVS partitioned datasets (PDSs).

## Operating Environments

TDCompress compression and decompression programs are compatible with these operating systems:

### TDCompress Operating Systems

#### **IBM Mainframes**

OS/390  
MVS/ESA (with LE/370 1.5)

#### **Midrange**

AIX 4.1 and higher  
DECUNIX  
DEC OpenVMS 7.1  
HPUX 10.01 and higher  
OS/400 V3R7 and higher  
SCO Open Server  
SUN Solaris 2.6

#### **PCs**

DOS 6.0  
OS/2 Warp  
Windows 3.1  
Windows 95/98  
Windows NT 4.0  
Windows2000  
WindowsXP

TDCompress processing options and programming interface provide flexibility for use in specialized file transfer environments.

# Chapter 1: Before You Begin

## Considerations

Issues to consider when performing cross-platform compression and security are the differences by which each machine stores and interprets data. This requires highly flexible compression software.

TDCompress meets this need by providing several run-time options for compressing and decompressing data, as well as providing an API that allows the user to control the compression/decompression process. Additional concerns are detailed below.

## ASCII-to-EBCDIC Translation

An obvious concern when transferring data between mainframes and workstations is the need to translate between the EBCDIC and ASCII character sets. TDCompress performs this translation when the ASCII run-time option is specified during compression. Although this option is valid for the mainframe, AS/400, and workstation compression programs, the actual translation takes place only when data is compressed or decompressed on the mainframe or AS/400 (for example, the EBCDIC machine). TDCompress uses the same ASCII-to-EBCDIC translation table as the IBM expEDite/PC product. The translation table can be customized at run time (see “Compressing on the Mainframe” on page 6-45 or “Decompressing on the Mainframe” on page 6-46).

## Handling Carriage Return/Line Feed Pairs

The **CRLF** run-time option is required in most cases to compensate for the differences in how records are stored on various computing platforms. MVS, OS/400, and VMS use highly structured, record-oriented I/O. Each record has a definite length that is either stored at the beginning of the record (variable-length records) or is contained in the definition of the file itself (fixed-length records). The various access methods use this information to read and write data one record at a time.

PC and UNIX workstations do not implement this record-oriented I/O. Files are processed as a continuous stream of bytes. The workstation leaves it up to the program to interpret the data as it sees fit. Sometimes a program is written so that it processes data in records of a fixed number of bytes. The program simply separates the data into fixed-length blocks for processing. More commonly, records on a DOS or OS/2 workstation are delimited by a carriage return/line feed combination of bytes (x'0D0A'). UNIX workstations use a line feed (x'0A') as the delimiter. This permits programs (and users) to determine where records end as well as to have variable-length records. High-level languages such as BASIC and C provide functions that read and write delimited data.

The user must know whether delimiters are present, or are needed, in the data that is compressed. When compressing data on the mainframe, the **CRLF** option adds a record separator character to the end of each record, which is replaced by the appropriate delimiter(s) when the data is decompressed on the workstation. When compressing data on the workstation, the **CRLF** option replaces the delimiters by a record separator character, which is used to write records of the correct length when the data is decompressed on the mainframe.

## Transferring Files Across Platforms

Text files compressed on an EBCDIC platform (MVS or AS/400) which are to be decompressed on an ASCII platform (UNIX or Windows) must be compressed with the **ASCII** and **CRLF** option and transmitted in binary mode by FTP. Binary files compressed with the **FILTER** option may be transmitted in either binary or ASCII mode.

## File Formats and Names

Another major difference between mainframes and workstations is the way files are structured and named. Sequential files on the mainframe can have names that are up to 44 characters in length. Mainframe files can also be stored as members of a partitioned data set (PDS), with each member name being up to eight characters long. The directory and file structure on a workstation is most analogous to a partitioned data set on the mainframe. A directory can be thought of as a PDS and each file in a directory can be thought of as a member. However, workstations allow more than eight characters for the file name. On the mainframe, file names are usually provided via Job Control Language (JCL) when a program is executed in batch. The workstation usually gets its file names from data entered at the command line or interactively by the user. TDCompress provides several methods for handling the different file structures and naming schemes. Each method is implemented via a combination of execution parameters and a possible user-supplied program. Various scenarios are discussed below.

## Upgrading to New Versions of TDCompress

Later releases of TDCompress can be used to decompress data compressed with an earlier release. However, data that is compressed using a later release cannot be decompressed using an earlier release of TDCompress. *In other words, the **DECOMP** programs are downward compatible, but the **COMPRESS** programs are not.*

The suggested method for software installation, after thoroughly testing the new release, is given below:

1. Install the new **DECOMP** program at the mainframe or central site.

2. Distribute the new **COMPRESS** or **DECOMP** programs to all remote sites (the new **DECOMP** programs can decompress data received from both the old and the new **COMPRESS** programs during the rollout period).
3. Use the **COMPRESS** version number printed for each decompressed segment in the Decompression Report to verify that all remote sites have installed the new **COMPRESS** or **DECOMP** programs.
4. Finally, install the new **COMPRESS** program at the mainframe or central site.

## Upgrading From Previous Versions

### Comm-Press 3.4.x and 4.4.x Migration Guide

Comm-Press 3.4.x and 4.4.x users need to be aware of the following compatibility issues with 3.0.1.

#### Secret Key Values

In 3.0.1, secret key values were entered as text on all platforms. For example, if the secret key was "MYSECRET" on MVS (an EBCDIC platform), it was also "MYSECRET" on a UNIX (ASCII) platform. While the text value is the same on each platform, the hexadecimal value is different.

M	Y	S	E	C	R	E	T
4D	59	53	45	43	52	45	54

Hexadecimal Values on an ASCII Platform

M	Y	S	E	C	R	E	T
D4	E8	E2	C5	C3	D9	C5	E3

Hexadecimal Values on an EBCDIC Platform

In order to allow for a larger set of secret keys, all keys in 3.4.x and 4.4.x are entered in hexadecimal format. This means that for the key shown above, either the hexadecimal values associated with the ASCII or EBCDIC key must be used on all platforms. If the key "MYSECRET" is used on an EBCDIC platform to encrypt data, the hexadecimal key value D4E8E2C5C3D9C5E3 must be used on ASCII platforms to decrypt the data.

3.4.x and 4.4.x users need to be aware that 3.0.1 versions of Comm-Press running on a ASCII platform translate all key values from ASCII to EBCDIC before attempting to decrypt data. This means that on an ASCII platform you must compress data with 3.4.1 or 4.4.1 using the hexadecimal key value D4E8E2C5C3D9C5E3 in order for the data to be decrypted with the key "MYSECRET" by 3.0.1 on any platform. If you take advantage of the larger key set in 3.4.x or 4.4.x by entering keys in hexadecimal, 3.0.1 users will be less likely to decrypt the data successfully.

## Secret Keys with Initialization Vectors

Secret keys in 3.4.x and 4.4.x are entered with an initialization vector to improve the randomness of the encryption. Version 3.0.1 does not support an initialization vector. If you want to encrypt data with 3.4.x or 4.4.x that is to be decrypted by 3.0.1, you must set the initialization vector of the secret key (bytes 25-32) to hexadecimal zeros (0000000000000000).

### SECURE

EDI data secured using the **SECURE** option in 4.4.x cannot be processed by 4.0.1.

### ENCRYPT

3.4.x and 4.4.x does not support this proprietary encryption option. **DES** or **DE3** should be used in its place. **Decomp** returns a return code of 52 if this option is used to encrypt a file in version 3.0.1 or 4.0.1. **Compress** returns a return code of 52 if this option is specified on a command line or in JCL.

### SAVEMODE

Data compressed with the parameter **SAVEMODE** cannot be decompressed by 3.0.1.

### User Exits

User exits are no longer supported in 3.4.x or 4.4.x. Use the enhanced TDCompress API to include the compression and encryption features into your custom applications

## Chapter 2: Securing Data Using TDCompress

TDCompress adheres to the ASC X12.58 subcommittee standards for applying security to X12-formatted EDI data. SxS and SxE segments are added to the EDI envelopes to carry bulk encryption/decryption information. SxA and SVA segments carry digital signature information. During decryption/decompression, the segments are removed from the data. SxA/SVA segments are retained if the KEEPSIGS execution option is specified.

For non-EDI data, TDCompress uses the S1S/S1E and S1A/SVA segments to carry encryption and digital signature information. As with EDI data, the segments are removed during decryption/decompression, unless the KEEPSIGS option is specified to retain the digital signature segments.

TDCompress writes security activity messages to log files. On the workstation, the log files are named `compress.log` and `decomp.log`. The logs are written to the current directory unless the **LOGPATH** command-line option or environment variable is set to a different directory. The AS/400 log files are named `COMPLOG` and `DCMPLOG` and are written to the current library. The **LOGPATH** command-line option can specify a different library. On MVS, the logs are written to the data sets defined via the `COMPLOG` and `DCMPLOG DD` statements. Each bulk encryption/decryption function and each digital signature function is logged. Errors are also logged.

## Execution Options for TDCompress Security Processing

TDCompress security processing is invoked by specifying the **SECURE** or **SECUREONLY** execution options. The security functions applied to the data are determined by the relationship records contained in the lookup run-time table. These relationships are defined by the TDManager Administrator. Other options may be specified to format the secured data for transmission or to provide lookup information for non-EDI data.

TDCompress Execution Options:

<b>ARCHIVE</b>	This <b>DECOMP</b> option is used to preserve specific information regarding an authenticated file. The information is stored in dynamically allocated files containing the input file information, and in the <code>decomp.log</code> . When using the <b>ARCHIVE</b> option, care must be taken to capture the information into a permanent location for later use.
<b>EDI</b>	This option is required when X12 or EDIFACT data is secured or unsecured. You must specify this option for both <b>COMPRESS</b> and <b>DECOMP</b> .

<b>DELIMIT[=n]</b>	The <b>COMPRESS</b> option causes the secured data to be delimited every <b>n</b> characters. The default for <b>n</b> is 40. This option may be required when transmitting secured data as a text file. It forces the <b>FILTER</b> option to be used if data is encrypted or compressed.
<b>KEEPSIGS</b>	This <b>DECOMP</b> option causes digital signature segments to be left in the unsecured data. The segments are normally removed from the data during decompression.
<b>LOGPATH=</b>	This option supplies the path (or AS/400 library) where <b>COMPRESS</b> and <b>DECOMP</b> write their log files. Workstation users can set the <b>LOGPATH</b> environment variable rather than use this option. If <b>LOGPATH</b> is not set, then the log files are written to the current directory.
<b>RECEIVER=</b>	This <b>COMPRESS</b> option can be used when non-EDI data is secured. It provides the value of the <b>RECEIVER</b> used to access the lookup run-time table. The <b>SENDER=</b> option must also be specified.
<b>RUNTIMEPATH=</b>	This option supplies the path (or AS/400 library) where <b>COMPRESS</b> and <b>DECOMP</b> can find the TManager run-time files. Workstation users can set the <b>RUNTIMEPATH</b> environment variable rather than use this option. If <b>RUNTIMEPATH</b> is not set, then the run-time files must be in the current directory.
<b>SECFILE</b>	This <b>COMPRESS</b> option may be used when securing non-EDI data. It identifies a file that contains statements defining user-supplied header records in the non-EDI data. TDCompress uses the header records to get the <b>SENDER</b> , <b>RECEIVER</b> , and <b>TRANSID</b> values used to access the lookup table. If the non-EDI data does not contain header records, then the <b>SECFILE</b> can provide the actual values for the <b>SENDER</b> , <b>RECEIVER</b> , and <b>TRANSID</b> to access the lookup table. See "Securing Non-EDI Data" on page 6-54 for more details on <b>SECFILE</b> .
<b>SECURE</b>	This <b>COMPRESS</b> option, or the <b>SECUREONLY COMPRESS</b> option, must be specified to invoke security processing. <b>SECURE</b> causes the Comm-Press program to use the run-time files from TManager to apply security to the data being compressed. With the <b>SECURE</b> option, if no security relationship is defined (for example, no record found in the lookup run-time file), then data is simply not secured, and no error is issued. This option is not required by <b>DECOMP</b> to unsecure the data.
<b>SECUREONLY</b>	This <b>COMPRESS</b> option, or the <b>SECURE COMPRESS</b> option, must be specified to invoke security processing. <b>SECUREONLY</b> causes the Comm-Press program to use the run-time files from TManager to apply security to the data being compressed. With the <b>SECUREONLY</b> option, if no security relationship is defined (for example, no record found in the lookup run-time file), then an error is issued, and processing stops. This option is not required by <b>DECOMP</b> to unsecure the data.

<b>SENDER=</b>	This <b>COMPRESS</b> option can be used when non-EDI data is secured. It provides the value of the <b>SENDER</b> that is used to access the lookup run-time table. The <b>RECEIVER=</b> option must also be specified.
<b>SQL(ssn,plan )</b>	This option can be used when securing data in an MVS-DB2 environment. If MVS-DB2 is the TDManager repository, then <b>COMPRESS</b> and <b>DECOMP</b> can directly access the tables for run-time information. This option specifies the DB2 subsystem name and plan used to access the TDManager database tables. Run-time files are not used when this option is specified.
<b>TRANSID=</b>	This <b>COMPRESS</b> option can be used when non-EDI data is secured. It provides the value of the <b>TRANSID</b> that is used to access the lookup run-time table. If no <b>TRANSID</b> is provided, then <b>COMPRESS</b> uses a default value of '*' (all non-EDI data). The <b>SENDER=</b> and <b>RECEIVER=</b> options must be specified.
<b>UNWRAP</b>	This option is only valid when the <b>EDI</b> option is specified. <b>UNWRAP</b> causes <b>COMPRESS</b> and <b>DECOMP</b> to begin each EDI segment on a new line.
<b>USEGS</b>	This <b>COMPRESS</b> option is only valid when the <b>EDI</b> option is specified. Normally the <b>SENDER</b> and <b>RECEIVER</b> values are taken from the X12 ISA segment. <b>USEGS</b> causes the GS02 and GS03 elements of the GS segment to be used instead.

## Extended Security Option Overview

The TDCompress Extended Security Option, used in conjunction with the TDManager product, provides complete, end-to-end security for X12-formatted EDI data, EDIFACT and non-EDI data. Security functions provided by TDCompress include bulk data encryption, using either the **DES**, Triple **DES** or **RC2** algorithms, authentication and digital signatures. At the heart of TDCompress security is public/private key technology licensed from RSA, Inc. An overview of secret and public/private key encryption as well as a description of their use in providing data security follows below.

## Overview of Secret Key Encryption

Traditional encryption algorithms rely on keys—a special piece of data to encrypt and decrypt data. These algorithms are known as symmetric, or secret, key algorithms because the same key is used to encrypt and decrypt data. The strength of an encryption algorithm is determined by how difficult it is to decrypt data without knowing the key used to encrypt it.

With strong algorithms such as **DES**, Triple **DES**, and **RC2**, it is not feasible to recover encrypted data without knowing the key. Therefore, attacks are performed to determine the key used during encryption. Strong algorithms are resistant to this type of attack; every possible key value would have to be attempted to determine the keys used to encrypt the data. With **DES**,  $2^{55}$  possible key values exist; a very large number to be attempted before successfully breaking the encrypted data. Triple **DES** has  $2^{167}$  possible keys.

Because of the difficulty in breaking strong encryption algorithms, attacks are made to intercept or steal keys. The secrecy of the keys is fundamental to maintaining the secrecy of the encrypted data. If trading partners are not diligent in protecting the keys when they are distributed and stored, then the data encrypted with those keys is also unprotected. For this reason, symmetric key algorithms require that secure channels and complex key distribution protocols be employed to share secret keys among trading partners. The trading partners themselves must then be relied upon to ensure the keys remain secret.

Public/private key technology enhances traditional encryption algorithms by providing easier and more secure use of secret keys. Electronic, or digital, signatures, possible only with public/private key technology, also provide authentication and assurance that data has not been tampered with after being secured. An overview of how public/private key technology enhances security follows.

## An Overview of Public/Private Key Encryption

With RSA's public/private key technology, each trading partner, or security participant, has two keys that are used during the security process. The public key is shared among all participants that wish to exchange secure data; while the private key is kept secret and is only known by the participant that owns it. Each public/private key pair is made up of two very large numbers that are the result of a complex mathematical function. Key pairs have the property such that when one key is used to encrypt data, only the other key in the pair can decrypt the data. For example, if the public key is used to encrypt, then only the private key will successfully decrypt. Key pairs also have the property such that even when the public key is known, it is computationally impossible to determine the private key. These two properties—the keys being inverses of each other and the mathematical difficulty of discovering the private key, are what make RSA's public/private key technology so useful in providing a higher degree of data security.

Since public keys are public, it is not necessary to secure them or take special precautions when distributing them among trading partners. In fact, the more widely known the public key, the more secure participants are from the possibility of an impostor trying to send false data by impersonating another trading partner. Private keys are never shared among security participants, so the possibility of them being intercepted is eliminated. The private keys are stored in an encrypted format by bTrade.com, so that even the owner of the private key does not normally know its actual value.

## Using Public/Private Key Technology to Encrypt Data

The drawback to public/private key encryption, is that encryption operations with the private key are relatively expensive in computational terms; so much so that bulk encryption of data using public/private keys is unfeasible due to the amount of time and computing resources required. Instead, public/private keys are used to enhance the implementation of faster secret key algorithms, such as :using **DES** or **RC2**, to perform bulk data encryption.

As discussed earlier, the problem with secret key algorithms is the need to maintain secrecy of the keys over a relatively long period of time and to ensure secure distribution of the keys among trading partners. Public/private key technology eliminates both of these concerns. With public/private key technology, a randomly generated value is used as the secret key for the bulk encryption algorithm. After being used to encrypt the data, this random key is encrypted with the recipient's public key and is sent along with the encrypted data. The recipient recovers the random key by decrypting it with his private key. Since the private key must be used to decrypt data encrypted by the public key and since private keys are known only by their owners, the recipient is the only person who can recover the random, secret key. The recipient then uses the recovered secret key to decrypt the received data.

## Digital Signatures and Authentication

Digital signatures, or assurances, are electronic signatures that can be applied to any electronic document. Digital signatures are more secure than actual handwritten signatures because they cannot be forged and they cannot be denied, or repudiated. This is due to the fact that the signer's private key is truly private. Forgery is not possible because only the signer has knowledge of his private key, and thus is the only person who can create his unique signature. Additionally, the signer cannot claim at some later date that the signature is invalid because anyone with knowledge of the signer's public key can verify the signature. If the signer's public key successfully verifies the signature, then the signer cannot deny that he created the signature.

A digital signature is created by first generating a hash, or message digest, of the electronic document being signed. The message digest is a large number that is the end product of running a hashing algorithm against the document. When a good hashing algorithm is used, it is statistically impossible for two documents to generate the same message digest. Thus, the digest is a unique mathematical representation of the document being signed. TDCompress uses the MD5 hashing algorithm to create message digests. Once the message digest is calculated, the digital signature is completed by encrypting the digest with the signer's private key.

The digital signature can be verified by anyone who knows the signer's public key. The verifier simply uses the same hashing algorithm used by the signer to regenerate the message digest. The verifier then decrypts the signature using the signer's public key. If the regenerated digest and the decrypted digest match, the signature is successfully verified. If the digests do not match, then either the document has been altered since the signature was created, or an impostor is trying to forge the digital signature.

TDCompress also uses digital signatures to provide data authentication. Authentication is used to verify that data has not been altered since it was sent. Authentication also ensures that the supposed sender, rather than an impostor, actually sent the data. From the previous discussion of digital signatures, one can see that both the objectives of authentication are provided by the signatures. Digital signatures add further security than authentication alone because they cannot be repudiated.

## Filtering

Filtering is used to convert compressed/encrypted data into a stream of printable characters. Filtering is useful when data is transmitted using protocols that do not support binary, or transparent, file transfers. Because filtered data consists only of printable characters, it may be transmitted as a normal text file without regard to cross-platform or network specific rules for data translation. While this ensures that the compressed/encrypted data will arrive at its destination intact, the drawback is that filtering increases the size of the compressed/encrypted data.

## Encrypting Data with TDCompress

You can encrypt compressed data by using one of the **COMPRESS** encryption options — **DES**, **DE3** or **RC2**. These options only provide bulk data encryption. More advanced security, such as key management and digital signatures using RSA public key technology, is available via the **SECURE** parameter in TDCompress Extended Security Option and the TDManager product.

Choose the **COMPRESS** encryption option depending on the degree of security required and, if the encrypted data is being exported to a foreign country, any export restrictions that apply to the encryption algorithm.

The **RC2** option applies the **RC2** encryption algorithm invented by RSA and provides the weakest encryption. It is exportable without restrictions. The **RC2** algorithm uses a 40-bit secret key.

The **DES** option applies the Data Encryption Standard algorithm and provides encryption. The U.S. government endorses the **DES** algorithm for encrypting non-classified data. The **DES** algorithm uses a 56-bit secret key.

The **DE3** option applies the **DES** algorithm three times with three different keys. This is known as the Triple **DES** algorithm and it provides strong encryption. Triple **DES** is not exportable without prior approval from the U.S. government. It uses a 168-bit secret key.

## Specifying the Secret Key and Initialization Vector

When using the **DES**, **DE3** or **RC2** options, the same secret encryption key must be provided during both compression and decompression. Additionally, an initialization vector must be provided to add randomness to the encrypted data. Both the key and initialization vector must be kept secret. Only the parties involved in the compression/encryption and decompression/decryption of the data should know them.

A secret key of the appropriate length depending on the algorithm chosen must be provided. **RC2** requires a 8-byte key (of which 40 key bits are used), **DES** requires an 8-byte key (56 key bits and 8 parity bits) and **DE3** requires a 24-byte key (168 key bits and 24 parity bits). The initialization vector for all algorithms is 8 bytes in length. Supply the key and initialization vector either on the command line or in a file.

## Supplying the Key and IV on the Command Line

Use the **KEY=** and **IV=** options to provide the values on the command line. Follow the equal signs in each option with the character representation of the hexadecimal values of the key and initialization vector. A command line example using the **DES** option follows.

```
compress input.fil output.fil des key=c5d4d4c1d5e4c5d3 iv=9d0a2300ed1c67f3
```

## Supplying the Key and IV in a File

To supply the secret key and initialization vector in a file, use a hexadecimal editor to create a file and enter their hexadecimal values.

- PC and UNIX workstation users must name the file `encrypt.key` and place it in the working directory when **COMPRESS** and **DECOMP** execute.
- OS/400 users create a file named **KEYIN**, which must be in the library list when **COMPRESS** and **DECOMP** execute.
- MVS users refer to the file via the **KEYIN DD** statement.
- VMS users must define the logical name **SY\$COMPRESS** to point to the drive and directory where **COMPRESS** and **DECOMP** will look for the file named `encrypt.key`.
- For all platforms, the file contains a single record. The first 24 bytes are reserved for the secret key. Bytes 25 through 32 are for the initialization vector.

An example `encrypt.key` file for use with the **DES** option follows. In the example, the key and initialization vector consist entirely of text bytes; however, no EBCDIC/ASCII translation of the values takes place. The bytes are treated as hexadecimal values. To enter a byte value, use a hexadecimal editor. Because **DES** requires an 8-byte key, spaces appear in bytes 9 through 24 of the secret key field. Column headings are shown for clarity.

<b>DONTTELL</b>	<b>ANYONE!!</b>
<b>1</b>	<b>25</b>

## Chapter 3: COMPRESS and DECOMP Options

The **COMPRESS** and **DECOMP** programs support several options that are used to invoke the programs' advanced features. Add options to the workstation command line or MVS and AS/400 **PARM** clause to invoke the features. Some features are common to all computer platforms supported by TDCompress and some features are platform specific. Tables listing the **COMPRESS** and **DECOMP** options and the computer operating platforms where they apply, are displayed in the following tables.

## Compress Options

### Common

	PC	UNIX	AS/400	MVS
APPEND	X	X	X	X
ASCII	X	X	X	X
AUTOEXT	X	X		
BISYNC	X	X	X	X
CRLF	X	X	X	X
DELETE	X	X		
DELIMIT	X	X		
DES	X	X	X	X
DE3	X	X	X	X
DIRNAME	X	X		
EDI	X	X	X	X
FILTER	X	X	X	X
IGNORE1A	X	X		
IV	X	X	X	X
KEY	X	X	X	X
LOGPATH	X	X	X	
LOOKUP	X	X	X	X
NOINFO	X	X	X	X
NOLOG	X	X	X	X
PF	X	X	X	X
QUIET	X	X	X	X
RC2	X	X	X	X
RECURSE	X	X		
SAVEMODE		X		
SIZE	X	X	X	X
SKIPWHITESPACE	X	X	X	X
SPEED	X	X	X	X
STDIN	X	X		
TRANTBL	X	X	X	X
TRLBLK			X	X

## TDCompress Extended Security Options

	<b>PC</b>	<b>UNIX</b>	<b>AS/400</b>	<b>MVS</b>
CCA				X
RECEIVER	X	X	X	X
RUNTIMEPATH	X	X	X	
SECFILE	X	X	X	X
SECURE	X	X	X	X
SECUREONLY	X	X	X	X
SENDER	X	X	X	X
SQL(ssn,plan)				X
TRANSID	X	X	X	X
USEGS	X	X	X	X

## Decompress Options

### Common

	<b>PC</b>	<b>UNIX</b>	<b>AS/400</b>	<b>MVS</b>
APPEND	X	X	X	X
AUTOEXT	X	X		
DELETE	X	X		
DIRNAME	X	X		
EDI	X	X	X	X
IV	X	X	X	X
KEY	X	X	X	X
LIST	X	X	X	X
LOGPATH	X	X	X	
LOWERCASE		X		
NOINFO	X	X	X	X
NOLOG	X	X	X	X
NOUNCOMP			X	X
ODATE	X			
QUIET	X	X	X	X
SAFE	X	X	X	X
SELECT	X	X	X	X
STDOUT	X	X		

TRANTBL	X	X	X	X
UNCOMP	X	X		
UNWRAP	X	X	X	X
USEINFO			X	

PC and UNIX workstation users can also set the **LOGPATH=** and **RUNTIMEPATH=** environment variables to set these locations globally.

### TDCompress Extended Security Option

	<b>PC</b>	<b>UNIX</b>	<b>AS/400</b>	<b>MVS</b>
ARCHIVE	X	X	X	X
CCA				X
KEEPSIGS	X	X	X	X
RUNTIMEPATH	X	X	X	
SECURECK	X	X	X	X
SQL(ssn,plan)				X

## Compress/Decompress Option Descriptions

<b>APPEND</b>	<p>This option commands <b>COMPRESS</b> and <b>DECOMP</b> to append data to the end of existing output files. The output files are created, if they do not exist. Appended compressed files can be separated into individual files during decompression. Note, that an implicit <b>APPEND</b> operation takes place when multiple input files are specified (via a file mask) but a single output file is produced. For example, the following command causes all files in the INDIR directory to be compressed into the single output file <code>COMP.OUT</code>:</p> <pre>Compress \indir\*. * \compwork\comp.out</pre>
<b>ARCHIVE</b>	<p>This option commands <b>DECOMP</b> to save specific information relating to an authenticated file to enable reprocessing. Related information is stored in a dynamically allocated file containing a copy of the input file as well as in the <code>decomp.log</code>. The syntax of this option is <b>ARCHIVE</b>—stores the dynamically allocated files into the current working directory or <b>ARCHIVE=path</b>—defines a directory path (or high-level qualifier) to be specified for file storage.</p>
<b>ARCHIVE=path</b>	See <b>ARCHIVE</b> .
<b>ASCII</b>	<p>This option causes <b>COMPRESS</b> to translate the data to ASCII or EBCDIC, if necessary, depending on the platform where the data is decompressed. The <b>ASCII</b> and <b>CRLF</b> options should always be used when compressing text files.</p>
<b>AUTOEXT [=n]</b>	<p>This option commands <b>COMPRESS</b> and <b>DECOMP</b> to automatically generate numeric extensions to name the output files. The optional value <b>n</b> specifies the number of digits to use in the extension. <b>n</b> may be in the range from one to nine. The default number of digits is three. The generated extensions start with two and continues to the highest number possible for the number of digits in the extension. All extensions are left-padded with zeros so they are <b>n</b> digits in length. The extensions are appended to the output name specified on the command line to create the full output file name. If the output name specified on the command line does not exist, then the name without an automatic extension, is used to name the first output file. Subsequent output files contain extensions. Existing output files are not overwritten. When decompressing, if the compressed files contain the original file name in the signature, the original names are used to name the output files unless the <b>NOINFO</b> option is specified. For example, the following command decompresses the input data into separate output files named <code>DCMP.OUT</code>, <code>DCMP.002</code>, <code>DCMP.003</code>, etc. The original file names are ignored.</p> <pre>decomp \indir\*. * \compwork\dcmp.out autoext noinfo</pre>

<b>BISYNC</b>	This option invokes a proprietary filter algorithm that prevents bytes within the compressed data from being misinterpreted as BISYNC control characters. Use of this option is not recommended. If your BISYNC communications software does not support transparency mode, use the <b>FILTER</b> option described below.
<b>CCA</b>	Optional Special Feature support. Refer to “Appendix D” on page 10-117 for information.
<b>CRLF</b>	This option commands <b>COMPRESS</b> to convert delimiter characters (for example, line feeds or carriage return/line feed pairs) into record separators. On AS/400 and MVS machines, <b>COMPRESS</b> inserts record separators at the end of each input record. During decompression, the delimiter characters that are appropriate for the target platform replace the record separators. On the PC and UNIX workstations, this option causes a x'1A' character to be treated as an end-of-file marker unless the <b>IGNORE1A</b> option is also chosen. The <b>ASCII</b> and <b>CRLF</b> options should always be used when compressing text files.
<b>DE3</b>	This option is not available when the software is exported unless special approval is received from the U.S. government. <b>DE3</b> invokes the Triple <b>DES</b> encryption algorithm. Triple <b>DES</b> is simply the <b>DES</b> algorithm executed three times with three different keys. A 24-byte encryption key (actually three 8-byte keys) must be supplied and the same key must be used to decompress the data. See “Encrypting Data with TDCompress” on page 2-18 for further details.
<b>DELETE</b>	This option causes <b>DECOMP</b> to delete the input files after successful decompression.
<b>DELIMIT=nn</b>	This option, in conjunction with the <b>FILTER</b> option, creates compressed data in delimited-text format. <b>COMPRESS</b> adds delimiters (for example, line feeds or carriage return/line feed pairs) after every <b>nn</b> characters of compressed output. The <b>FILTER</b> option is automatically invoked if this option is used.
<b>DES</b>	This option invokes the Data Encryption Standard—U.S. government standard encryption algorithm. An 8-byte encryption key must be supplied and the same key must be used to decompress the data. See “Encrypting Data with TDCompress” on page 2-18 for further details.

<b>DIRNAME</b>	<p>For <b>COMPRESS</b>, this option stores the full path of the input file, including the directory, in the compressed output. Default action is to store only the file name. For <b>DECOMP</b>, this option uses the input file path names stored in the compressed data to build an output path for the decompressed files. The input file path names are concatenated to the output path specified on the command line, or to the current directory if no output path was specified, in order to generate the actual path name for the decompressed files. If the resulting output path does not exist, it is automatically created.</p> <p>This option is not valid in combination with the EDI parameter because directory information is not stored by <b>COMPRESS</b> processing.</p>
<b>EDI</b>	<p>This option may be used when X12, EDIFACT, UN/EDI, or UCS EDI data is compressed. The header and trailer records that mark the start and end of an EDI envelope are left uncompressed in the output file. Use of this parameter also ensures that the compressed data does not contain any characters that could be misinterpreted as EDI control characters. If the <b>EDI</b> parameter is used to compress the data, then the <b>EDI</b> parameter must also be used to decompress the data.</p>
<b>FILTER</b>	<p>This option invokes the <b>FILTER</b> algorithm described in RFC 1113 to convert the compressed data from binary to text format. Filtered data is always transmitted as text. Use this option when the data communication environment does not allow transparent data transmission. <b>FILTER</b> is also required when the output data contains a combination of compressed and uncompressed data, as is the case when the <b>EDI</b> or <b>PF</b> options are used, and the data is transmitted between unlike computer platforms. For example, EDI data on a PC that is sent to an AS/400 should be compressed with the <b>EDI</b> and <b>FILTER</b> options. The resulting compressed file is then sent to the AS/400 as a text file.</p>
<b>IGNORE1A</b>	<p>This option, when used with the <b>CRLF</b> option, causes <b>COMPRESS</b> to not treat any x'1A' characters as end-of-file markers. Default processing when the CRLF option is chosen is to stop reading input when a x'1A' character is read.</p>
<b>IV=iv</b>	<p>This option, when used with one of the encryption options <b>DES</b>, <b>DE3</b> or <b>RC2</b>, provides the initialization vector for the encryption algorithm. See "Encrypting Data with TDCompress" on page 2-18 for further details</p>
<b>KEEPSIGS</b>	<p>This option is only valid when decompressing secured data. Digital signatures present in the secured data are verified and then removed from the output files by default. This option keeps digital signatures in the output. See "Securing Data Using TDCompress" on page 2-12 for further details.</p>
<b>KEY=key</b>	<p>This option, when used with one of the encryption options <b>DES</b>, <b>DE3</b> or <b>RC2</b>, provides the key for the encryption algorithm. See "Encrypting Data with TDCompress" on page 2-18 for further details</p>

<b>LIST</b>	<p>This parameter causes <b>DECOMP</b> to print a listing of the compressed input with each compressed segment categorized by its size and file information (if present). <b>LIST</b> also checks each compressed segment for data integrity. Decompression is not performed when <b>LIST</b> is specified.</p> <p>This option is not valid in combination with the EDI parameter because the information is not stored by <b>COMPRESS</b> processing.</p>
<b>LOGPATH=path</b>	<p>This option gives the path or AS/400 library where the compression and decompression log files are written. The default is to write the log files in the current directory or AS/400 library. The names of the log files are compress.log and decomp.log. Workstation users can also set the <b>LOGPATH=</b> environment variable to set the location globally.</p>
<b>LOOKUP</b>	<p>This option causes <b>COMPRESS</b> to read the <b>CPLOOKUP</b> file to determine whether to compress EDI data. See "Securing Formatted EDI Data" on page 6-53 for further details.</p>
<b>LOWERCASE</b>	<p>This option causes <b>DECOMP</b> to convert all file names stored in the compressed data to lower case before naming the decompressed output files.</p>
<b>NOINFO</b>	<p>For <b>COMPRESS</b>, this option overrides the default storage of the input file name and date in the compressed output. This information is normally used to name the output files, if they are decompressed on a PC or UNIX workstation. For <b>DECOMP</b>, this option causes any directory information stored in the compressed files to be ignored when naming the decompressed output files.</p> <p>This option is not valid in combination with the EDI parameter because the information is not stored by <b>COMPRESS</b> processing.</p>
<b>NOLOG</b>	<p>This option suppresses the creation of the compression and decompression log files.</p>
<b>NOUNCOMP</b>	<p>This option causes the MVS and AS/400 decompression programs to ignore uncompressed data in the input. The default is to write uncompressed data to the output file <i>as is</i>.</p>
<b>ODATE</b>	<p>This option causes decompressed output files to retain the creation date and time of the original compressed input files.</p> <p><b>ODATE</b> is only valid in true DOS mode.</p>

<b>PF=filename</b>	<p>This option is used when certain input records are to be left uncompressed. On PCs and UNIX workstations, the input data must be delimited for this option. Replace filename with the fully qualified name of a parameter file that contains information about the records that are to be left uncompressed. Each record in the parameter file contains a character string, its starting position in the input records, and its length. <b>COMPRESS</b> examines each input record to see if one of the character strings occurs in the specified location. If a match is found, then the record is left uncompressed in the output file. The parameter file can contain as many records as necessary to provide all the character strings used to identify uncompressed records. The format of the parameter file records is:</p> <pre>&lt;starting position&gt; &lt; length &gt; &lt;character string&gt;</pre> <p>Each parameter file record must end with a record delimiter appropriate to the computer platform (for example, a line feed character on UNIX, or a carriage return/line feed pair on a PC). As an example, if records that contain the string <b>HEADER</b> starting in column 10 and records that contain the string <b>TRAILER</b> starting in column 1 are to be left uncompressed, then the parameter file will contain the following two records:</p> <pre>10 6 HEADER 1 7 TRAILER</pre> <p>This option is not valid in combination with the <b>EDI</b> parameter.</p>
<b>QUIET</b>	This option suppresses all output messages from <b>COMPRESS</b> and <b>DECOMP</b> .
<b>RC2</b>	This option invokes the <b>RC2</b> encryption algorithm. This algorithm is exportable to foreign countries with no special authorization from the U.S. government. A 8-byte encryption key must be supplied because it must also be used to decompress the data. See "Encrypting Data with TDCompress" on page 2-18 for further details.
<b>RECEIVER=rcvr</b>	This option may be used to provide the name of the security receiver when securing data. <b>RECEIVER</b> is only valid when the <b>SECURE</b> option is also specified. See "Securing Data Using TDCompress" on page 2-12 for further details.
<b>RECURSE</b>	This option causes <b>COMPRESS</b> to travel down the subdirectory tree and compress all files that match the input file mask. The <b>DIRNAME</b> option is automatically invoked when this option is specified.
<b>RUNTIMEPATH=path</b>	This option may be used to provide the path where the security run-time files are located. <b>RUNTIMEPATH</b> is only valid when the <b>SECURE</b> option is also specified. Workstation users can also set the <b>RUNTIMEPATH=</b> environment variable to set the location globally. See "Securing Data Using TDCompress" on page 2-12 for further details.
<b>SAFE</b>	This option causes <b>DECOMP</b> to create a reject file containing data that fails decompression or decryption. Only valid data is written to the output file(s).

<b>SAVEMODE</b>	This option causes UNIX and VMS file modes to be saved in the compression signature. The modes are restored on <b>DECOMP</b> .
<b>SECFILE=filename</b>	This option provides the name of the security definition file that may be required when securing non-EDI data. This option is only valid when the <b>SECURE</b> option is specified. See "Securing Data Using TDCompress" on page 2-12 for further details.
<b>SECURE</b>	This option invokes the advanced security features of TDCompress Extended Security Option. This is the lenient form of invoking security. If no security relationship is defined (for example, no record found in the lookup run-time file), then the data is not secured, and no error is issued. Contrast this with the <b>SECUREONLY</b> option. See "Securing Data Using TDCompress" on page 2-12 for further details.
<b>SECURECK</b>	With this option <b>DECOMP</b> ensures that only decrypted data is written to the output file(s). Input data that is not encrypted is written to a reject file.
<b>SECUREONLY</b>	This option invokes the advanced security features of TDCompress Extended Security Option. This is the strict form of invoking security. If no security relationship is defined (for example, no record found in the lookup run-time file), then an error is issued and processing stops. "Securing Data Using TDCompress" on page 2-12 for further details.
<b>SELECT ( )</b>	<p>This option allows selective decompression of compressed segments. The segments must contain embedded file name information (included by default when compressing on an AS/400 or workstation). Specify the exact file names of the segments to decompress in the parentheses that follow the <b>SELECT</b> parameter. Separate multiple file names with a comma. If multiple segments with the same file name are in the input file, then by default, the first segment is decompressed. Override this default action by adding a colon and relative sequence number to the file name. An example follows:</p> <pre>decomp infile \work select(file.txt.1)</pre> <p>The above example reads the compressed archive file named INFILE. It decompresses the first occurrence of the compressed file named FILE.TXT. \WORK is the directory where the decompressed file is written.</p> <p>This option is not valid in combination with the <b>EDI</b> parameter because the filename is not stored by <b>COMPRESS</b> processing.</p>
<b>SENDER=sndr</b>	This option is used to provide the name of the security sender when securing data. <b>SENDER</b> is only valid when the <b>SECURE</b> option is specified. See "Securing Data Using TDCompress" on page 2-12 for further details.

<b>SIZE</b>	This option creates the smallest possible compressed output at the expense of processing time.
<b>SKIPWHITESPACE</b>	This option is valid only in conjunction with the EDI parameter. <b>SKIPWHITESPACE</b> causes invalid characters in EDI segments (for example, carriage return/line feed characters) to be skipped. This option is useful when the transmission software inserts the invalid characters and causes <b>COMPRESS</b> and <b>DECOMP</b> to process the EDI data incorrectly.
<b>SPEED</b>	This option decreases processing time at the expense of compressed output size. In many cases the savings in processing time make up for the slightly larger compressed output.
<b>SQL(ssn,plan)</b>	This option can be used when securing data in an MVS-DB2 environment. If MVS-DB2 is the TDManager repository, then <b>COMPRESS</b> and <b>DECOMP</b> can directly access these tables for run-time information. This option specifies the DB2 subsystem name and plan used to access the TDManager database tables. Run-time files are not used when this option is specified.  Default:   ssn=DSN   plan=COMPSQL
<b>STDIN</b>	This option causes <b>COMPRESS</b> to read its input from <b>STDIN</b> rather than from disk files.
<b>STDOUT</b>	This option causes <b>DECOMP</b> to write its output to <b>STDOUT</b> rather than to disk files. Specify the <b>QUIET</b> parameter in addition to <b>STDOUT</b> to keep messages from appearing in the data file.
<b>TRANSID=trid</b>	This option is used to provide the name of the security transaction ID when securing data. <b>TRANSID</b> is only valid when the <b>SECURE</b> option is specified. See "Securing Data Using TDCompress" on page 2-12 for further details.
<b>TRANTBL</b>	This option provides the name of an ASCII/EBCDIC translation table to use when compressing and securing data.  <b>TRANTBL</b> when used on an ASCII workstation does not translate data being compressed. It would only affect hash processing for authentication. Yes or No overrides the <b>CPLOOKUP</b> table. Filename gives the name of an actual translation table. Transupdate is supported on MVS.
<b>TRLBLK</b>	This option, when used with the <b>CRLF</b> option, causes trailing blanks in MVS and AS/400 records to be removed from the compressed data. Default <b>CRLF</b> processing keeps trailing blanks.

<b>UNCOMP</b>	<p>This option signals <b>DECOMP</b> that the input files contain valid uncompressed data in addition to compressed data. <b>DECOMP</b> will copy the uncompressed data to the output files as it decompresses. If this option is not specified, then any data that occurs between the end of a compressed segment and the beginning of the next compressed segment are assumed to be pad characters and are ignored.</p> <p>This option is not valid in combination with the EDI parameter, since all data outside the EDI envelope is ignored and passed as-is by default.</p>
<b>UNCOMPSIG</b>	<p>This option tells <b>DECOMP</b> to expect a signature at the end of the file during a session.</p>
<b>UNWRAP</b>	<p>This option is valid only in conjunction with the <b>EDI</b> option. When both the <b>EDI</b> and <b>UNWRAP</b> options are specified, <b>COMPRESS</b> and <b>DECOMP</b> split <b>EDI</b> segments so that each segment begins on a new record.</p>
<b>USEGS</b>	<p>This option is only valid when the <b>SECURE</b> and <b>EDI</b> options are also specified. <b>USEGS</b> causes the GS02 and GS03 elements of the X12 GS segment to be used as the security sender and receiver. See “Securing Data Using TDCompress “ on page 2-12 for further details.</p>
<b>USEINFO</b>	<p>This option is used to name decompressed members on an AS/400. <b>DECOMP</b> does not normally create members when decompressing. All decompressed data is appended into the output file specified on the command line. <b>USEINFO</b> uses the file names stored in the compressed data to create members in the output file.</p>

## Chapter 4: Key Generation and TDManager Run-time Files (TDCompress Extended Security Option)

To take advantage of the advanced security features of TDCompress, the user must have an RSA public/private key pair and security run-time files from TDManager.

Public/private key pairs are generated in one of several ways:

- As part of the TDManager Trading Partner Registration
- Using the TDAccess software
- Using the **GENKEYS** utility distributed with TDCompress

Refer to the *TDManager User Guide* or the *TDAccess User Guide* for instructions on using these products.

The **GENKEYS** utility is used when batch key generation is required. **GENKEYS** requires an input configuration file, named **easyacc.ini**, that is produced by the TDManager product. Additional input parameters provide **GENKEYS** with the name of the passphrase location file, the path for the input **easyacc.ini** file, and user-provided randomization data. **GENKEYS** is provided on the AS/400, MVS and workstation platforms.

The **IMPORT** utility is a batch program that installs the TDManager security run-time files. Once the files are imported, the user can begin securing data with their trading partners.

### RSA Key Generation

The **GENKEYS** utility reads the **EASYACC** configuration file from TDManager and creates two output files — **CERTREQ** and **PRIVKEY**. The **PRIVKEY** file created by **GENKEYS** is a permanent key file and must be retained. The **CERTREQ** file contains the portion of the key that the TDManager certifies. The TDManager Administrator issues the security run-time files required to transmit secure data. Refer to the respective platforms for installation instructions.

## TDManager Run-time Files

When using the **ARCHIVE** option of **DECOMP**, the TDManager Administrator retains participant records for affected files. These participants may have their certificates revoked, which consequently inactivates them. However, the participant must remain in the tables, so that all certificates are available for archived information.

TDManager is a Windows application that provides complete key and trading partner relationship management. It creates run-time files that are used by TDCompress to secure data transmitted between trading partners. The run-time files created by TDManager and used by TDCompress to secure data are the certificate, private key and symmetric key files, as well as the lookup and participant tables. See the *TDManager User Guide* for complete details on using TDManager. A brief description of the run-time files and how they are used by TDCompress follows.

### The Certificate Run-time File

The certificate file contains the public keys of all the trading partners who wish to exchange secure data. The public keys are stored in a standard format — ANSI X.509 — called a certificate to which TDManager and TDCompress adhere. Certificates contain the distinguished name of the public key owner as well as a copy of the public key, and the starting and ending validity dates of the key. The entire certificate is digitally signed by a trusted authority (for example, the TDManager) who certifies and issues the public key.

For workstation users, the certificate file is called cert.fil. The **RUNTIMEPATH** command-line option or environment variable points to the directory where the file is stored.

For MVS users, the certificate file is referenced via the PUBLKEYS DD statement.

For AS/400 users the certificate file is called CERT and must be accessible via the library list.

### The Private Key Run-time File

The private key file contains the private keys of local security participants that originate and send secure data from the site where TDCompress is run. Private keys are never shared among trading partners.

For workstation users, the private key file is called private.fil. The **RUNTIMEPATH** command-line option or environment variable points to the directory where the file is stored.

For MVS users, the private key file is referenced via the **PRIVKEYS** DD statement, or may be accessed directly from DB2 tables using the SQL parameter.

For AS/400 users, the private key file is called **PRIVATE** and must be accessible via the library list.

The private keys are stored so unauthorized persons cannot view them. Each private key is encrypted with a random passphrase when the key is generated to ensure the secrecy of the private key. However, TDCompress needs the passphrase at run time to decrypt the private key and perform security functions. (For example, creating a digital signature or decrypting an encrypted **DES** or **RC2** key). Depending on options specified when the private key is generated, the passphrase may either be stored with the private key itself, or it may be supplied in an external file. Storing the passphrase with the private key is a convenience feature. This is recommended if the private key file is protected against unauthorized viewing and copying. If the passphrase is stored in an external file, then the name of the file containing the passphrase is specified when the keys are generated. The passphrase file can be on removable media, such as a diskette, that is stored in a secure location when not in use. For MVS users, the passphrase file should be RACF or ACF2 protected. The passphrases are encrypted with a static encryption key to guard against casual viewing. However, it is the user's responsibility to employ appropriate procedures to protect the passphrase files from unauthorized access.

## The Symmetric Key Run-time File

The symmetric key file is only present when interoperating with older security software such as Sterling Software's Dataguard product. The symmetric key file contains the secret **DES** keys used to authenticate and encrypt data according to the symmetric key standards of X12.58.

For workstation users, the symmetric key file is called `symkey.fil`. The **RUNTIMEPATH** command-line option or environment variable points to the directory where the file is stored.

For MVS users, the lookup table is referenced via the SYMKEY DD statement or may be accessed directly from DB2 tables using the SQL parameter.

For AS/400 users, the symmetric key file is called SYMKEY. It must be accessible via the library list.

## The Lookup Table Run-time File

The lookup table contains records that define security options used between trading partners. Lookup table records contain keyword/value combinations that define the sender, receiver, transaction type and security options for each trading partner relationship.

For workstation users, the lookup table is called **CPLOOKUP.TBL**. The **RUNTIMEPATH** command-line option or environment variable points to the directory where the file is stored.

For MVS users, the lookup table is referenced via the **CPLOOKUP** DD statement or may be accessed directly from DB2 tables using the SQL parameter.

For AS/400 users, the lookup table is called **CPLOOKUP**. It must be accessible via the library list.

TDCompress looks up security options based on **sender**, **receiver** and **transaction type** values. If a match is found, then the security options specified for that trading partner relationship are used to secure the data.

## The Participant Table Run-time File

The participant table contains information used to build EDI security segments. The table is optional, however, TDCompress uses default values when none are specified or when the file is not present.

For workstation users, the participant table is called **partic.tbl**. The **RUNTIMEPATH** command-line option or environment variable points to the directory where the file is stored.

For MVS users, the participant table is referenced via the **PARTIC DD** statement or may be accessed directly from DB2 tables using the SQL parameter.

For AS/400 users, the participant table is called **PARTIC**. It must be accessible via the library list.

# Chapter 5: SECFILE Keywords By Platform

## MVS

### SECFILE Keywords for Data with User-Defined Headers

**COMPRESS** can get the **SENDER**, **RECEIVER** and **TRANSID** values from user-defined headers in the data. **SECFILE** keywords must be coded to inform **COMPRESS** how to recognize a header record as well as where the **SENDER**, **RECEIVER** and **TRANSID** values are in the header. All **SECFILE** keywords are coded using a **KEYWORD(VALUE)** syntax. Note that the first parenthesis must immediately follow the keyword with no intervening spaces. **SECFILE** keywords can be coded on multiple lines in the file, however, keywords and values cannot be split across lines. A list of the valid **SECFILE** keywords along with a short description of their use follows below.

Parameter	Description
HEADERLITERAL()	Specifies the unique literal that identifies header records
HEADERSTART()	Specifies the starting column of the header literal
SENDERSTART()	Specifies the starting column of the sender field
SENDERLENGTH()	Specifies the length of the sender field (max of 15)
RECEIVERSTART()	Specifies the starting column of the receiver field
RECEIVERLENGTH()	Specifies the length of receiver field (max of 15)
TRANSIDSTART()	Specifies the starting column of the transaction field (optional)
TRANSIDLENGTH()	Specifies the length of the transaction field (optional; max of 3)

The parameters are discussed in detail, based on a sample file containing two header records followed by data records. The example file is shown below:

column:	1	10	20	30
data	HEADER	SNDR01	RCVR01	PO
	data record			
	data record			
	HEADER	SNDR02	RCVR02	PO
	data record			
	data record			

In the above example, the header literal begins in column one of the header records. The sender field begins in column ten and is six characters in length. The **RECEIVER** field begins in column twenty and is also six characters in length. The **TRANSID** field begins in column thirty with a two character length.

A user-defined header record must contain a literal value somewhere in the record for **COMPRESS** to be able to distinguish it as a header. The **SECFILE** keywords **HEADERLITERAL()** and **HEADERSTART()** tell **COMPRESS** what the header literal is as well as where to find it on the header record. For the above example these keywords would be coded as follows:

```
HEADERLITERAL(HEADER)    HEADERSTART(1)
```

Next, the fields containing the **SENDER**, the **RECEIVER** and the **TRANSID** must be identified to **COMPRESS**. The **SENDERSTART()**, **SENDERLENGTH()**, **RECEIVERSTART()**, **RECEIVERLENGTH()**, **TRANSIDSTART()** and **TRANSIDLENGTH()** keywords accomplish this. For the above example these parameters would be coded as follows:

```
SENDERSTART(10)    SENDERLENGTH(6)
RECEIVERSTART(20)  RECEIVERLENGTH(6)
TRANSIDSTART(30)   TRANSIDLENGTH(2)
```

The set of keywords should be ended with a semicolon. So, for example, the entire **SECFILE** would be coded as follows:

```
HEADERLITERAL(HEADER)    HEADERSTART(1)
SENDERSTART(10)           SENDERLENGTH(6)
RECEIVERSTART(20)         RECEIVERLENGTH(6)
TRANSIDSTART(30)          TRANSIDLENGTH(2);
```

A header record need not contain all three of the **SENDER**, **RECEIVER** and **TRANSID** values. Undefined values default to '\*'.

## SECFILE Keywords for Data with No Headers

When non-EDI data does not contain any user-defined headers, **SECFILE** keywords simply provide the **SENDER**, **RECEIVER** and **TRANSID** values for **COMPRESS** to use to search the lookup table. This implies that the file can only be sent to a single recipient. The **SECFILE** parameters used to provide the information are:

Parameter	Description
<b>SENDER()</b>	Specifies the sender value
<b>RECEIVER()</b>	Specifies the receiver value
<b>TRANSID()</b>	Specifies the transaction value

For example, to secure a file being sent from user **SNDR01** to recipient **RCVR01** with a **TRANSID** of **PO**, the **SECFILE** would be coded as follows:

```
SENDER(SNDR01)    RECEIVER(RCVR01)  TRANSID(PO);
```

All three **SENDER**, **RECEIVER** and **TRANSID** values need not be specified. Undefined values default to '\*'.

## MVS Example for Securing Non-EDI Data

```
//JOBNAME  JOB   (ACCOUNT INFO),'USER DATA',CLASS=A,MSGCLASS=X
//COMP      EXEC  PGM=COMPRESS,PARM='SECURE SECFILE=DD:SECFILE',
//          REGION=1024K
//STEPLIB   DD    DSN=your.load.library,DISP=SHR
//SYSPRINT  DD    SYSOUT=*
//COMPLOG   DD    SYSOUT=*
//DATAIN    DD    DSN=input.file.to.compress,DISP=OLD
//DATAOT    DD    DSN=compressed.output.file,DISP=(,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1,1)),
//          LRECL=xxxxxx,BLKSIZE=xxxxxx,RECFM=xx
//PUBLKEYS  DD    DSN=secmgr.cert.fil,DISP=SHR
//PRIVKEYS  DD    DSN=secmgr.private.fil,DISP=SHR
//SYMKEY    DD    DSN=secmgr.symkey.fil,DISP=SHR
//CPLOOKUP  DD    DSN=secmgr.lookup.tbl,DISP=SHR
//PARTIC    DD    DSN=secmgr.partic.tbl,DISP=SHR
//SECFILE   DD    DSN=secfile.parms.defining.user.headers,DISP=SHR
```

# PC

## SECFILE Keywords for Data with User-Defined Headers

**COMPRESS** can get the **SENDER**, **RECEIVER** and **TRANSID** values from user-defined headers in the data. **SECFILE** keywords must be coded that inform **COMPRESS** how to recognize a header record, as well as where the **SENDER**, **RECEIVER** and **TRANSID** values are in the header. All **SECFILE** keywords are coded using a 'KEYWORD(VALUE)' syntax. Note that the first parenthesis must immediately follow the keyword with no intervening spaces. **SECFILE** keywords can be coded on multiple lines in the file; however, keywords and values cannot be split across lines. A list of the valid **SECFILE** keywords along with a short description of their use follows.

Parameter	Description
HEADERLITERAL()	Specifies the unique literal that identifies header records
HEADERSTART()	Specifies the starting column of the header literal
SENDERSTART()	Specifies the starting column of the sender field
SENDERLENGTH()	Specifies the length of the sender field (max of 15)
RECEIVERSTART()	Specifies the starting column of the receiver field
RECEIVERLENGTH()	Specifies the length of receiver field (max of 15)
TRANSIDSTART()	Specifies the starting column of the transaction field (optional)
TRANSIDLENGTH()	Specifies the length of the transaction field (optional; max of 8)
HEADERDROP()	(Default - N) - Reads values from header before dropping header from data stream. (Y) - Keeps headers in data stream.
HEADERCLEAR()	(Default - Y) - Leaves headers in the clear within the data stream. (N) - Compresses headers within data stream.

The parameters are discussed in detail, based on a sample file containing two header records followed by data records. The example file is shown below:

column:	1	10	20	30
data	HEADER	SNDR01	RCVR01	PO
	data record			
	data record			
	HEADER	SNDR02	RCVR02	PO
	data record			
	data record			

In the above example, the header literal begins in column one of the header records. The **SENDER** field begins in column ten and is six characters in length. The **RECEIVER** field begins in column twenty and is six characters in length. The **TRANSID** field begins in column thirty and is two characters long.

A user-defined header record must contain a literal value somewhere in the record for **COMPRESS** to be able to distinguish it as a header. The **SECFILE** keywords **HEADERLITERAL()** and **HEADERSTART()** tell **COMPRESS** what the header literal is, as well as where to find it on the header record. For the above example these keywords would be coded as follows:

```
HEADERLITERAL(HEADER)    HEADERSTART(1)
```

Next, the fields containing the **SENDER**, the **RECEIVER** and the **TRANSID** must be identified to **COMPRESS**. The **SENDERSTART()**, **SENDERLENGTH()**, **RECEIVERSTART()**, **RECEIVERLENGTH()**, **TRANSIDSTART()** and **TRANSIDLENGTH()** keywords accomplish this. For the above example these parameters would be coded as follows:

```
SENDERSTART(10)          SENDERLENGTH(6)
RECEIVERSTART(20)        RECEIVERLENGTH(6)
TRANSIDSTART(30)         TRANSIDLENGTH(2)
```

The set of keywords should be ended with a semicolon. So, for the example, the entire **SECFILE** would be coded as follows:

```
HEADERLITERAL(HEADER)    HEADERSTART(1)
SENDERSTART(10)          SENDERLENGTH(6)
RECEIVERSTART(20)        RECEIVERLENGTH(6)
TRANSIDSTART(30)         TRANSIDLENGTH(2);
```

A header record need not contain all three of the **SENDER**, **RECEIVER**, and **TRANSID** values. Undefined values default to '\*'.

## SECFILE Keywords for Data with No Headers

When non-EDI data does not contain any user-defined headers, **SECFILE** keywords simply provide the **SENDER**, **RECEIVER**, and **TRANSID** values for **COMPRESS** to use to search the lookup table. This implies that the file can only be sent to a single recipient. The **SECFILE** parameters used to provide the information are:

Parameter	Description
<b>SENDER()</b>	Specifies the sender value
<b>RECEIVER()</b>	Specifies the receiver value
<b>TRANSID()</b>	Specifies the transaction value

For example, to secure a file being sent from user **SNDR01** to recipient **RCVR01** with a **TRANSID** of **PO**, the **SECFILE** would be coded as follows:

```
SENDER(SNDR01)    RECEIVER(RCVR01)    TRANSID(PO);
```

All three of the **SENDER**, **RECEIVER** and **TRANSID** values need not be specified. Undefined values default to '\*'.

## PC Example for Securing Non-EDI Data

The following command is an example of how to compress and secure a file containing non-EDI data. In the example, the file named DATA.FIL contains the data to be compressed/secured. It contains user-defined header records. The input file will be compressed/secured into the DATA.CMP file in the COMPWORK directory. The file named HEADER.DEF contains the **SECFILE** keywords that describe the user-defined headers in the input file. The run-time files must be in the current directory, unless the **RUNTIMEPATH** environment variable has been set. The 'compress.log' file is written to the current directory, unless the **LOGPATH** environment variable has been set.

```
COMPRESS DATA.FIL \COMPWORK\DATA.CMP SECURE SECFILE=HEADER.DEF
```

# Chapter 6: MVS Platform

## MVS Installation

The TDCompress MVS software is distributed either on a PC-DOS diskette, CD-ROM, or a 3480 tape cartridge. Instructions for installing the TDCompress MVS software are shown on the following pages.

### Using a diskette or CD-ROM

The disk files are compressed and self-extracting. Move the file to a PC file system and double-click the filename to begin the installation application.

The resulting files are created using the TSO **TRANSMIT** command. To install the files on MVS, see this procedure:

1. Transfer the resulting files from the PC to a mainframe using a 3270 file transfer program or FTP. The files must be transferred in binary format to a file that has been defined with the attributes as RECFM=FB, LRECL=80, and BLKSIZE=3120. An example is listed below:

```
ftp 180.138.16.2                <= connect to MVS/OS390
220 User (none): userid          <= enter USERID
331 Enter password:              <= enter PASSWORD
230 USERID logged on.
ftp> bin                         <= binary mode
200 Representation type is binary IMAGE.
ftp> quote site recfm=fb lrecl=80 blksize=3120
200 Site command was accepted
ftp> put loadlib.441 'user.compress.file' rep
200 PORT subcommand request successful.
125 Storing data set user.compress.file
250 Transfer completed successfully.
ftp> quit                        <= disconnect
```

2. Once the files are on the mainframe, issue the TSO **RECEIVE** command to unload the files into a partitioned data set (PDS). See the next example.

```
RECEIVE INDA('USER01.UPLOAD.FILE')
```

This PDS is in MVS LOADLIB format with a RECFM (record format) of U and a BLKSIZE of 6144. The PDS does not need to be pre-allocated because the TSO **RECEIVE** command allocates it during the unload. If you wish to unload the file into an existing library, it must have the same RECFM and BLKSIZE attributes illustrated in the example above.

3. The **RECEIVE** command issues a prompt before it unloads the file, respond with the name of the PDS where the TDCompress modules are installed. See the following example:

```
Enter restore parameters or DELETE or END +    <=== prompt from RECEIVE
DA('USER01.LIBRARY')                          <=== type PDS name
```

## Using a 3480 tape cartridge

The distribution tape contains two files—a library containing the MVS load modules and the sample JCL containing routines to execute the **COMPRESS** and **DECOMP** programs.

Below is an example of MVS JCL to unload the TDCompress distribution tape:

```
//UNLOAD    JOB    (ACCOUNT INFO),'user info',CLASS=A,MSGCLASS=X
//COPY1     EXEC   PGM=IEBCOPY
//SYSPRINT  DD     SYSOUT=*
//INDD1     DD     DSN=CMMPRESS.LOADLIB,DISP=(OLD,PASS),
//           UNIT=TAPE,VOL=SER=CMTAPE,LABEL=(1,SL)
//OUTDD1    DD     DSN=USER.LOADLIB,DISP=(NEW,CATLG),UNIT=SYSDA,
//           SPACE=(CYL,(2,2,5)),BLKSIZE=6144,RECFM=U
//SYSIN     DD     *
//          COPY I=INDD1,O=OUTDD1
//*
//COPY2     EXEC   PGM=IEBCOPY
//SYSPRINT  DD     SYSOUT=*
//INDD2     DD     DSN=CMMPRESS.SAMPLIB,DISP=OLD,
//           UNIT=TAPE,VOL=SER=CMTAPE,LABEL=(2,SL)
//OUTDD2    DD     DSN=USER.SAMPLIB,DISP=(NEW,CATLG),
//           UNIT=SYSDA,SPACE=(TRK,(5,5,5)),BLKSIZE=3120,
//           LRECL=80,RECFM=FB
//SYSIN     DD     *
//          COPY I=INDD2,O=OUTDD2
//
```

## Binding the DB2 Plan (Extended Security Option)

If MVS-DB2 is used as the TDManager repository, the DB2 tables can be accessed directly through TDCompress for run-time information. (For example relationships, certificates, keys, etc.), specify the SQL parameter on the **COMPRESS/DECOMP** execution.

Remember that the DB2 plan used to access the TDManager tables must be bound. The bind must be repeated following any change to the DB2 tables or to the **COMPRESS** or **DECOMP** modules that access them. The BINDJCL member in the **SAMPLIB** contains a sample job that may be used to bind the plan. Refer to the comments in BINDJCL for details.

# Installing the TDManager Run-time Files

## Using IMPORT on MVS

1. To complete the security configuration, install the run-time files generated by the TDManager.
  - a. Or optionally refer to the TDManager DB2 tables directly.
    - i. The run-time files are distributed as single, compressed and encrypted files, which are installed by the **IMPORT** utility.
    - ii. Sample IMPORT JCL is distributed in the **SAMPLIB PDS**. Instructions for running the job and installing your run-time files are included in the sample JCL.

MVS users must upload **export.fil** to a sequential data set using an FTP that supports binary file transfers. No ASCII/EBCDIC or carriage return/line feed processing should be performed during the file transfer. Once the file is uploaded, the **IMPORT** job, supplied in the TDCompress **SAMPLIB** can be run to decompress the three run-time files. The **IMPORT** job contains two steps. The first step decompresses the run-time files contained in **export.fil**. The dynamic allocation feature of the MVS **DECOMP** program is used to allocate the three run-time files. The second step processes the decompressed private key file and creates the external passphrase files. The passphrase files are dynamically allocated using the exact data set names entered in TDManager.

Sequential passphrase files must not exist prior to running the **IMPORT** job. They are created via dynamic allocation. However, if passphrases are stored as members of a partitioned data set, the PDS must exist before the **IMPORT** job is run. Dynamic allocation then creates the individual members that hold the passphrases.

Installation of the run-time files completes the security configuration. See “Securing Data Using TDCompress” on page 2-12 for details on securing files using TDCompress.

## Mainframe Operation

TDCompress contains two mainframe load modules named **COMPRESS** and **DECOMP**. The **COMPRESS** and **DECOMP** load modules perform data compression and decompression on an IBM mainframe running the MVS operating systems. The compressed data can be encrypted to ensure that data is kept confidential.

The **COMPRESS** module reads a sequential or partitioned data set, identified via the **DATAIN DD**, and writes a compressed sequential dataset, identified via the **DATAOT DD**. EBCDIC-to-ASCII translation may be performed during compression and carriage return/line feed delimiters may be added to the input records. Special options are also included for compressing EDI-formatted data. Details for executing **COMPRESS** on the mainframe are given in “Compressing on the Mainframe” on page 6-45.

**DECOMP** reads a compressed sequential dataset identified via the **DATAIN DD** and writes a decompressed sequential or partitioned data set, identified via the **DATAOT DD**. **DECOMP** is designed to handle an input data set that contains compressed data from multiple remote sites. The compressed data from each site has a signature in the first 16 bytes of the first compressed record (see “Appendix B” on page 10-93 for details). This signature notifies **DECOMP** of the start of a group of compressed records. Each group of compressed records is decompressed and written to the output data set. **DECOMP** accepts uncompressed data mixed with compressed data in the input data set. Any record that does not belong to a group of compressed records is simply written unchanged to the output data set. Details for executing **DECOMP** on the mainframe are given in “Decompressing on the Mainframe” on page 6-46.

## Compressing on the Mainframe

TDCompress supports fixed or variable, blocked or unblocked records. Spanned record formats are not supported. Input is specified via the **DATAIN DD** statement and output is specified via the **DATAOT DD** statement. The minimum output record length is 16 bytes for fixed records and 20 bytes for variable records. Both input and output records may be up to the maximum record length allowed by the operating system. **DATAIN** and **DATAOT** may specify different record formats and record lengths. **COMPRESS** uses values set in the DCB to determine the record formats and lengths.

Execution parameters invoke various built-in features that prepare the data for transmission and decompression. Parameters are provided to create platform-independent compressed files (for example, **ASCII** and **CRLF**) as well as allow the compression of specialized data (for example EDI).

If the **ASCII** option is specified, **COMPRESS** translates the input from EBCDIC to **ASCII**. **COMPRESS** uses a default table to perform the translation. The default character translations can be modified at execution time. A listing of the default table as well as instructions for overriding the defaults are provided in “**Error! Reference source not found.**” **Error! Bookmark not defined.**

**COMPRESS** parameters are specified via the **PARM** field on the execute statement. Multiple parameters must be separated by at least one space. See “Compress Options” on page 3-21 for further information.

Sample MVS JCL to compress mainframe data follows:

```
//JOBNAME  JOB  (ACCOUNT INFO),'USER DATA',CLASS=A,MSGCLASS=X
//COMP     EXEC PGM=COMPRESS,PARM='ASCII CRLF',
//          REGION=1024K
//STEPLIB  DD   DSN=your.load.library,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//DATAIN   DD   DSN=input.file.to.compress,DISP=OLD
//DATAOT   DD   DSN=compressed.output.file,DISP=(,CATLG),
//          UNIT=SYSDA,SPACE=(CYL,(1,1)),
//          LRECL=xxxxxx,BLKSIZE=xxxxxx,RECFM=xx
//KEYIN    DD   *   <=== required if DES, DE3 or RC2 parms specified
DONTTELL          ANYONE!!
//
```

## Decompressing on the Mainframe

**DECOMP** supports fixed or variable, blocked or unblocked records. Spanned record formats are not supported. Input is specified via the **DATAIN DD** statement and output is specified via the **DATAOT DD** statement. The minimum input record length is 16 bytes for fixed records and 20 bytes for variable records. There is no minimum output record length and there is no maximum length for either the input or output data sets. **DATAIN** and **DATAOT** may specify different record formats and record lengths. **DECOMP** uses values set in the DCB to determine the record formats and lengths. The input data set may contain both compressed and uncompressed data. **DECOMP** uses the 16-byte signature at the start of each segment of compressed data to identify data that must be decompressed. Uncompressed data is simply copied to the output data set as is.

**DECOMP** processes a sequential dataset as input. It writes either sequential or PDS output datasets. **DECOMP** determines the DSORG of the input and output data sets at execution time.

When the output dataset is a PDS and the input file contains multiple compressed segments, such as when multiple workstation files are compressed into a single file, each segment is decompressed into a separate output PDS member. Decompressed PDS output member names are either copied from the original input PDS member names, are made the same as the first 8 bytes of the file name stored in the compressed data, or are generated by **DECOMP**. Generated names are of the form CPnnnnnn, where **nnnnnn** is a six-digit number that starts at 000001 and is incremented by one for each output member.

The dynamic allocation feature of **DECOMP** provides an alternative to splitting multiple compressed segments into individual files. With this feature, **DECOMP** allocates a separate sequential output dataset for each compressed segment encountered in the input. Further information and restrictions regarding use of the dynamic allocation feature are in the “Using Dynamic Allocation with DECOMP” on page 6-48.

If the input was compressed with the **ASCII** option, **DECOMP** translates the decompressed output from ASCII to EBCDIC. **DECOMP** uses a default table to translate data from ASCII to EBCDIC. The default character translations can be modified at execution time. A listing of the default table as well as instructions for overriding the defaults are provided in “**Error! Reference source not found.**” **Error! Bookmark not defined.**

Selective decompression of compressed input segments is provided via the **SELECT** control statement. **SELECT** statements are provided by the user via the **SYSIN DD**. Further information on selective decompression is provided in “**Error! Reference source not found.**” **Error! Bookmark not defined.**

**DECOMP** parameters are specified via the **PARM** field on the execute statement. Multiple parameters must be separated by at least one space. Refer to “**Error! Reference source not found.**” **Error! Bookmark not defined.** for further information.

Sample MVS JCL to decompress mainframe data (without dynamic allocation) follows:

```
//JOBNAME JOB (ACCOUNT INFO), 'USER DATA', CLASS=A, MSGCLASS=X
//DECOMP EXEC PGM=DECOMP,
// PARM='EDI SQL(DSN,COMPPLAN)',
// REGION=1024K
//STEPLIB DD DSN=your.load.library, DISP=SHR
//SYSPRINT DD SYSOUT=*
//DATAIN DD DSN=compressed.input.file, DISP=OLD
//DATAOT DD DSN=decompressed.output.file, DISP=(,CATLG),
// UNIT=SYSDA, SPACE=(CYL,(1,1)),
// LRECL=xxxxxx, BLKSIZE=xxxxxx, RECFM=xx
//KEYIN DD * <=== required if the compressed data is encrypted
DONTTELL ANYONE!!
//
```

## Using ARCHIVE with DECOMP

**DECOMP** has the ability to collect specific information required to process secure files at a later time. When the **ARCHIVE** keyword is specified, a copy of the input file is placed into the directory path (or high-level qualifier) specified by **ARCHIVE=**. When **ARCHIVE** is specified alone, the default directory is the current working directory. **DECOMP** can dynamically allocate the copy of the input. Dynamic allocation parameters, which are allocated in the same way as the dynamically allocated output dataset(s), may be overridden using **DYNARCH DD \***. Please refer to “Using Dynamic Allocation with DECOMP” below, and to the **SAMPLIB (ARCHIVE)** member for specific information and requirements.

When using **ARCHIVE**, the user is responsible for preserving the archive file copy, as well as the **decomp.log** for future processing. Additionally, TDManager participant records for both sender and receiver must not be deleted from the database. These participants may have their certificates revoked, in order to disable their keys, but they must remain as inactive participants to preserve the certificate information.

## Using Dynamic Allocation with DECOMP

**DECOMP** has the ability to dynamically allocate its output dataset(s). This feature is useful when the input contains multiple compressed segments, as is the case when multiple workstation files are compressed into a single file for transmission. Dynamic allocation can be used to separate the files when they are decompressed at the mainframe. To use dynamic allocation, the compressed segments must contain file name information (this is the default when files are compressed on the workstation). The embedded file names are used to build the names of the dynamically allocated datasets. The **DATAOT DD** statement should be omitted when using dynamic allocation.

The dynamic allocation feature is invoked when the user provides dynamic allocation control statements via the **SYSIN DD**. The control statements describe the format and characteristics used to dynamically allocate the output datasets. Each control statement contains one or more keywords that define the output datasets. Most keywords also require a value to be supplied. These parameters are entered in the form:

**KEYWORD=value**

Keywords can be entered in any order and on as many input statements as necessary. Separate keywords with at least one blank or comma. Do not enter keywords past column 72 and do not split a keyword/value pair across multiple input statements.

The DSN, LRECL and BLKSIZE keywords are required. Other keywords may be provided as needed. Valid keywords and their values follow. The default values are underlined:

DSN=xxxxxxxx.xxxxxxxxx.xxxxxxxxx	high-level dsn qualifiers (31 chars max)
DISP= <u>NEW</u>   MOD   OLD	initial dataset disposition
NDISP= <u>CATLG</u>   KEEP   DELETE	normal dataset disposition (normal EOJ)
CDISP= <u>CATLG</u>   KEEP   DELETE	conditional dataset disposition (ABEND)
TRACKS   <u>CYLINDERS</u>   BLKLEN=nnn	allocation units
PRIMARY=nnn   <u>2</u>	primary allocation amount (in above units)
SECONDARY=nnn   <u>2</u>	secondary allocation amount
<u>RELEASE</u>   NORELEASE	release unused space at close
UNIT=xxxxxxxx   <u>SYSALLDA</u>	unit for allocation device type
UNITCOUNT=nn   <u>1</u>	number of units to allocate
BLKSIZE=nnnnn   <u>0</u>	output block size (must be non-zero)
LRECL=nnnnn   <u>0</u>	output record length (must be non-zero)
RECFM= <u>FB</u>   VB	output record format

**DECOMP** appends the file name from the compressed input file as additional qualifiers to the high-level qualifiers supplied in the DSN parameter.

Sample MVS JCL to decompress mainframe data using dynamic allocation follows (note the absence of the **DATAOT DD** statement):

```
//JOBNAME JOB (ACCOUNT INFO),'USER DATA',CLASS=A,MSGCLASS=X
//DECOMP EXEC PGM=DECOMP,
// REGION=1024K
//STEPLIB DD DSN=your.load.library,DISP=SHR
//SYSPRINT DD SYSOUT=*
//DATAIN DD DSN=compressed.input.file,DISP=OLD
//KEYIN DD * <=== required if the compressed data is encrypted
DONTTELL ANYONE!!
//
//SYSIN DD *
DSN=HIGHLVL.QUAL
DISP=NEW NDISP=CATLG CDISP=DELETE
CYLINDERS PRIMARY=10 SECONDARY=5 RELEASE UNIT=SYSDA
BLKSIZE=4100 LRECL=4096 RECFM=VB
//
```

## Modifying the Default Translation Tables at Run Time

TDCompress contains default EBCDIC-to-ASCII and ASCII-to-EBCDIC translation tables. The user can modify the tables, at run time, if the defaults are not appropriate for the data being processed. Before modifying the tables, the user must understand when TDCompress performs the actual translation to/from EBCDIC/ASCII.

All translation between the EBCDIC and ASCII character sets is performed on the EBCDIC host (for example MVS). Translation from EBCDIC to ASCII occurs during compression when the ASCII execution parameter is specified; while translation from ASCII to EBCDIC occurs during decompression if the ASCII parameter was supplied during data compression (on any platform). This means that the MVS COMPRESS program contains the default EBCDIC-to-ASCII translation table and the MVS DECOMP program contains the default ASCII-to-EBCDIC translation table.

The default tables can be modified by supplying alternate translation values for particular bytes via the **TRANUPD DD**. The alternate translation values are specified in the following format:

**natural value=translated value**

In the above format, natural value is the EBCDIC or ASCII value to be translated from and the translated value is the corresponding ASCII or EBCDIC value to be translated to. Only one pair of natural/translated values may appear on each record read from the **TRANUPD DD**. For example, to translate the EBCDIC character A to the ASCII character zero and the EBCDIC character zero to the ASCII character A, the following must be included in the **COMPRESS JCL**:

```
//TRANUPD DD *
C1=30          EBCDIC 'A' to ASCII '0'
F0=41          EBCDIC '0' to ASCII 'A'
/*
```

To translate the ASCII character A to the EBCDIC character zero and the ASCII character zero to the EBCDIC character A, the following must be included in the **DECOMP JCL**:

```
//TRANUPD DD *
41=F0          ASCII 'A' to EBCDIC '0'
30=C1          ASCII '0' to EBCDIC 'A'
/*
```

As many statements as necessary may be provided via the **TRANUPD DD** to modify the translation tables.

Below is the default EBCDIC-to-ASCII translation table (displayed in hexadecimal format) used by the MVS COMPRESS program. To locate the substitute ASCII value for a particular EBCDIC byte, use the left-most digit of the EBCDIC byte as the row number in the table and the right-most digit of the EBCDIC byte as the column number. The cell where the row and column intersect contains the ASCII value. For example, EBCDIC byte X'C1' is translated to ASCII X'41' and EBCDIC X'4D' becomes ASCII X'28'. The table can be modified at run time by supplying update records via the TRANUPD DD, as described previously.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	80	81	82	83	84	8E	87	8F	88	89	8A	8B	8C	8D	86	7F
3	90	91	97	93	94	95	96	9C	98	99	9A	9B	9D	85	9E	92
4	20	A0	A1	A2	A3	A4	A5	A6	A7	A8	D5	2E	3C	28	2B	7C
5	26	A9	AA	AB	AC	AD	AE	AF	B0	B1	21	24	2A	29	3B	5E
6	2D	2F	B2	B3	B4	B5	B6	B7	B8	B9	E5	2C	25	4F	3E	3F
7	BA	BB	BC	BD	BE	BF	C0	C1	C2	60	3A	23	40	27	3D	22
8	C3	61	62	63	64	65	66	67	68	69	C4	C5	C6	C7	C8	C9
9	CA	6A	6B	6C	6D	6E	6F	70	71	72	CB	CC	CD	CE	CF	D0
A	D1	7E	73	74	75	76	77	78	79	7A	D2	D3	D4	5B	D6	D7
B	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	5D	E6	E7
C	7B	41	42	43	44	45	46	47	48	49	E8	E9	EA	EB	EC	ED
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	EE	EF	F0	F1	F2	F3
E	5C	9F	53	54	55	56	57	58	59	5A	F4	F5	F6	F7	F8	F9
F	30	31	32	33	34	35	36	37	38	39	FA	FB	FC	FD	FE	FF

Table 1: Default EBCDIC-to-ASCII Translation Table

Below is the default ASCII-to-EBCDIC translation table (displayed in hexadecimal format) used by the MVS **DECOMP** program. To locate the substitute EBCDIC value for a particular ASCII byte, use the left-most digit of the ASCII byte as the row number in the table and the right-most digit of the ASCII byte as the column number. The cell where the row and column intersect contains the EBCDIC value. For example, ASCII byte X'41' is translated to EBCDIC X'C1' and ASCII X'28' becomes EBCDIC X'4D'. The table can be modified at run time by supplying update records via the **TRANUPD DD**, as described previously.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
1	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	40	5A	7F	7B	5B	6C	50	7D	4D	5D	4C	4E	6B	60	4B	61
3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	2F
8	20	21	22	23	24	3D	2E	26	28	29	2A	2B	2C	2D	25	27
9	30	31	3F	33	34	35	36	32	38	39	3A	3B	37	3C	3E	E1
A	41	42	43	44	45	46	47	48	49	51	52	53	54	55	56	57
B	58	59	62	63	64	65	66	67	68	69	70	71	72	73	74	75
C	76	77	78	80	8A	8B	8C	8D	8E	8F	90	9A	9B	9C	9D	9E
D	9F	A0	AA	AB	AC	4A	AE	AF	B0	B1	B2	B3	B4	B5	B6	B7
E	B8	B9	BA	BB	BC	6A	BE	BF	CA	CB	CC	CD	CE	CF	DA	DB
F	DC	DD	DE	DF	EA	EB	EC	ED	EE	EF	FA	FB	FC	FD	FE	FF

Table 2: Default ASCII-to-EBCDIC Translation Table

## Using GENKEYS on MVS

The **GENKEYS** utility reads the **EASYACC** configuration file from TManager and creates two output files — **CERTREQ** and **PRIVKEY**. Sample **GENKEYS** JCL is distributed in the **SAMPLIB** PDS. Instructions for running the job and creating your keys are included in the sample JCL.

The **PRIVKEY** file created by **GENKEYS** is a permanent key file and must be retained. The **CERTREQ** file contains the portion of the key that must be certified by the TManager. The TManager Administrator issues the security run-time files required to transmit secure data. See “Installing the TManager Run-time Files” on page 6-44 for a description of the run-time files and details on how to install them.

## Securing Formatted EDI Data

To secure formatted-EDI data, both the EDI and either the **SECURE** or **SECUREONLY** run-time options must be specified during compression. The run-time files created by TManager, as well as any required passphrase files, must be available.

When TDCompress processes the formatted data, the **SENDER** and **RECEIVER** fields from the ISA or UNB segment, along with the group ID from the GS segment, or the transaction type from the ST or UNH segment, are used to perform a search in the lookup table. If a record exists in the lookup table with matching **SENDER**, **RECEIVER**, and **TRANSACTION** values, then the security options specified on the record are used to secure the data in the EDI envelope.

### MVS Example for Securing Formatted EDI Data

```
//JOBNAME JOB (ACCOUNT INFO),'USER DATA',CLASS=A,MSGCLASS=X
//COMP EXEC PGM=COMPRESS,PARM='EDI SECUREONLY',
// REGION=1024K
//STEPLIB DD DSN=your.load.library,DISP=SHR
//SYSPRINT DD SYSOUT=*
//COMPLOG DD SYSOUT=*
//DATAIN DD DSN=input.X12.file.to.compress,DISP=OLD
//DATAOT DD DSN=compressed.output.file,DISP=(,CATLG),
// UNIT=SYSDA,SPACE=(CYL,(1,1)),
// LRECL=xxxxxx,BLKSIZE=xxxxxx,RECFM=xx
//PUBLKEYS DD DSN=secmgr.cert.fil,DISP=SHR
//PRIVKEYS DD DSN= secmgr.private.fil,DISP=SHR
//SYMKEY DD DSN= secmgr.symkey.fil,DISP=SHR
//CPLOOKUP DD DSN= secmgr.cplookup.tbl,DISP=SHR
//PARTIC DD DSN= secmgr.partic.tbl,DISP=SHR
```

## Unsecuring Formatted EDI Data

To unsecure formatted-EDI data, the **EDI** option must be specified during decompression. The TDManager run-time files as well as any required passphrase files must also be available.

When TDCompress decompresses the secured EDI data, it processes security segments to recover the random bulk encryption key. It then decrypts and decompresses the data and verifies digital signatures. Error messages are created if any security features fail.

### MVS Example for Unsecuring Formatted EDI Data

```
//JOBNAME JOB (ACCOUNT INFO), 'USER DATA', CLASS=A, MSGCLASS=X
//COMP EXEC PGM=DECOMP, PARM='EDI',
// REGION=1024K
//STEPLIB DD DSN=your.load.library, DISP=SHR
//SYSPRINT DD SYSOUT=*
//DCMPLOG DD SYSOUT=*
//DATAIN DD DSN=input.X12.file.to.decompress, DISP=OLD
//DATAOT DD DSN=decompressed.output.file, DISP=(,CATLG),
// UNIT=SYSDA, SPACE=(CYL,(1,1)),
// LRECL=xxxxxx, BLKSIZE=xxxxxx, RECFM=xx
//PUBLKEYS DD DSN=secmgr.cert.fil, DISP=SHR
//PRIVKEYS DD DSN=secmgr.private.fil, DISP=SHR
//SYMKEY DD DSN=secmgr.symkey.fil, DISP=SHR
//PARTIC DD DSN=secmgr.partic.tbl, DISP=SHR
```

## Securing Non-EDI Data

To secure non-EDI data, the **SECURE** or **SECUREONLY** option must be specified during compression. In addition, the **SECFILE** or **SENDER/RECEIVER/TRANSID** options must be specified during compression. The run-time files created by TDManager as well as any required passphrase files must also be available.

Non-EDI data does not contain an independently-defined standard header like the ISA segment present in X12 data, for **COMPRESS** to get **SENDER**, **RECEIVER** and **TRANSID** values. These values can be provided by the user via the **SENDER**, **RECEIVER** and **TRANSID** command-line options or keywords in the **SECFILE**.

If the non-EDI data contains user-defined header records containing the **SENDER**, **RECEIVER** and **TRANSID** values, **SECFILE** keywords can be used to describe the proprietary headers to **COMPRESS**. Details for coding the **SECFILE** are given on page 5-36.

## Unsecuring Non-EDI Data

To unsecure non-EDI data, no special run-time options need to be specified during decompression. However, the run-time files created by TDManager as well as any required passphrase files must be available.

When TDCompress decompresses the secured non-EDI data, it processes the S1S segments to recover the random bulk encryption key; decrypts and decompresses the data; and verifies digital signatures. Error messages are created when security features fail.

### MVS Example for Unsecuring Non-EDI Data

```
//JOBNAME  JOB  (ACCOUNT INFO),'USER DATA',CLASS=A,MSGCLASS=X
//COMP      EXEC PGM=DECOMP,REGION=1024K
//STEPLIB   DD   DSN=your.load.library,DISP=SHR
//SYSPRINT   DD   SYSOUT=*
//DCMPLOG    DD   SYSOUT=*
//DATAIN     DD   DSN=input.file.to.decompress,DISP=OLD
//DATAOT     DD   DSN=decompressed.output.file,DISP=(,CATLG),
//           UNIT=SYSDA,SPACE=(CYL,(1,1)),
//           LRECL=xxxxxx,BLKSIZE=xxxxxx,RECFM=xx
//PUBLKEYS   DD   DSN=secmgr.cert.fil,DISP=SHR
//PRIVKEYS   DD   DSN=secmgr.private.fil,DISP=SHR
//SYMKEY     DD   DSN=secmgr.symkey.fil,DISP=SHR
//PARTIC     DD   DSN=secmgr.partic.tbl,DISP=SHR
```

## Compressing Mainframe Files

A mainframe sequential file is, by default, compressed to an output sequential file. However, many times the sequential file is composed of logically separate groups of records or segments. For example, a host-based electronic mail system is used to create messages for several remote users. At some time during the day, the messages are extracted from the e-mail system into a sequential file. Each group of messages for a particular remote user, or possibly each individual message, is a logically separated segment since they must be transmitted to different destinations. Default **COMPRESS** processing is not suitable for this situation, since the sequential input file cannot simply be read from beginning to end and compressed to a single sequential output file. For these situations, bTrade.com provides an API that allows a user-written program to control the compression process.

TDCompress processes both sequential and partitioned data sets as input. The output must be a sequential dataset. bTrade.com determines the DSORG of the input and output data sets at execution time.

When the input is a PDS, member names are stored as the file name in each compressed member. The entire PDS is compressed to the sequential output dataset for transmission as a single file. The decompression program splits the compressed data back into separate files using the original PDS member name as the file name or member name.

## Modifying the Default Translation Tables

Comm-Press contains a default EBCDIC-to-ASCII translation table. The user can modify the table at run time if the default translation is not appropriate.

All translation between the EBCDIC and ASCII character sets is performed on the EBCDIC host (for example, MVS or AS/400). Translation from EBCDIC to ASCII occurs during compression when the ASCII execution parameter is specified. Translation from ASCII to EBCDIC occurs during decompression if the ASCII parameter was supplied when the data was compressed.

The default table is replaced by providing a file with the desired EBCDIC-to-ASCII translation. The DD name of the file is specified via the **TRANTBL=** parameter. The file must contain 256 pairs of characters. Each pair of characters represent one hexadecimal ASCII byte. The position of each pair is the EBCDIC value to translate and the value of each pair is the ASCII value to translate to. For example, the first pair of characters gives the ASCII translation for the EBCDIC value x'00', the second pair gives the ASCII translation for the EBCDIC value x'01' and so on. The last pair gives the ASCII translation for the EBCDIC value x'FF'. Spaces or commas can be used to separate the pairs for readability.

Below is the TDCompress default EBCDIC-to-ASCII translation table as it would be entered in a file that is pointed to by the **TRANTBL DD**:

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
80 81 82 83 84 8E 87 8F 88 89 8A 8B 8C 8D 86 7F
90 91 97 93 94 95 96 9C 98 99 9A 9B 9D 85 9E 92
20 A0 A1 A2 A3 A4 A5 A6 A7 A8 D5 2E 3C 28 2B 7C
26 A9 AA AB AC AD AE AF B0 B1 21 24 2A 29 3B 5E
2D 2F B2 B3 B4 B5 B6 B7 B8 B9 E5 2C 25 5F 3E 3F
BA BB BC BD BE BF C0 C1 C2 60 3A 23 40 27 3D 22
C3 61 62 63 64 65 66 67 68 69 C4 C5 C6 C7 C8 C9
CA 6A 6B 6C 6D 6E 6F 70 71 72 CB CC CD CE CF D0
D1 7E 73 74 75 76 77 78 79 7A D2 D3 D4 5B D6 D7
D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 5D E6 E7
7B 41 42 43 44 45 46 47 48 49 E8 E9 EA EB EC ED
7D 4A 4B 4C 4D 4E 4F 50 51 52 EE EF F0 F1 F2 F3
5C 9F 53 54 55 56 57 58 59 5A F4 F5 F6 F7 F8 F9
30 31 32 33 34 35 36 37 38 39 FA FB FC FD FE FF

```

The following MVS JCL fragment shows how to use the **TRANTBL=** parameter:

```
//DECOMP    EXEC  PGM=DECOMP,PARM='TRANTBL=EBC2ASC',  
//          REGION=1024K  
//EBC2ASC   DD    DSN=CP.CNTL(EBC2ASC),DISP=SHR
```

# Chapter 7: VMS Platform

## Installation

### TK7

The distribution tape (TK70 format) contains the COMPRESS.EXE and DECOMP.EXE programs. The tape is created with the VMS backup command, and a file saveset of comp.bck. Use the backup command to copy the two files into the directory that you want to use.

### Disk

If you received a 3.5-inch disk, transfer the three files on the disk to the VMS system using kermit with a file type option of fixed binary for the file COMPRESS.TLB. Execute the EXTRACTX.COM program, which expands the COMPRESS.TLB file into the **COMPRESS** and **DECOMP** executables.

To use the compress software, the following symbols must be added to either the user's login.com or to the system wide sylogin.com:

```
$ COMP*RESS == "$ disk:[compress_directory] COMPRESS.EXE
$ DECOMP*RESS == "$ disk:[compress_directory] DECOMP.EXE
```

Where disk and compress\_directory are the disk and directories in which TDCompress was installed.

The '\*' in the symbol names indicate the minimum characters that may be entered to have the name of the program recognized.

Note: The \$ before the disk and directory name is critical, without it the program will not accept parameters.

## VMS Operation

TDCompress consists of two programs — COMPRESS.EXE and DECOMP.EXE — which perform data compression and decompression on VMS version 5.1 or greater, or on OpenVMS 7.1 or greater.

The COMPRESS.EXE and DECOMP.EXE programs must be copied to and executed from a directory on a hard drive. Fully qualified file names are supported (for example, file paths may be included when specifying file names). Wildcard characters may be used to specify input and output file masks.

Both COMPRESS.EXE and DECOMP.EXE are invoked by symbols defined in the user's login procedure or in the global login procedure. The symbols specify the fully qualified program names, including drive and directory, and must be used to pass command-line parameters. The following are the suggested symbols to define:

```
$ COMP*RESS == "$ disk:[compress_directory]COMPRESS.EXE
$ DECOMP*RESS == "$ disk:[compress_directory]DECOMP.EXE
```

Where disk and compress\_directory are the disk and directories in which TDCompress was installed.

The '\*' in the symbol names indicate the minimum characters that may be entered to have the name of the program recognized.

The **SAVEMODE** option preserves the RMS file attributes of the original file as well as the permissions with two exceptions. System permission is replaced by Owner permission, and those with Write permission also have Delete permission when the file is decompressed. This is caused by a compatibility issue between VMS and UNIX permissions.

File attributes such as Record format and attributes, and file organization are preserved in the decompressed file.

When multiple input files are compressed on VMS, the user has the option to create individual compressed output files or append the compressed files into a single output file (the files are separated again when decompressed). Default COMPRESS.EXE processing stores directory information with the compressed output files, including the file name and extension. This information is used to name the files when they are decompressed. If the directory information is not needed, then COMPRESS.EXE options may be used to override the default processing. **RECFM** and **LRCL** options are not required to create an archive containing like files.

Files of different types such as executables and stream files may be stored in the same compressed archive using the **APPEND** option. Use the **SAVEMODE** option to preserve the file attributes, otherwise the files may be unusable. Use the **RECFM=V** and **LRCL=32767** options during compression in order to later **APPEND** two archives containing different file attributes. Do not use the options **CRLF** and **ASCII** with binary files because this produces an unusable file after decompression.

DECOMP.EXE allows multiple compressed files or segments to exist in the input file. If file name and extension information is available in the compressed segments, then the segments are separated into individually decompressed output files using the name and extension information embedded in the compressed segment. This default processing may be overridden by the user via DECOMP.EXE options.

## Compressing on VMS

The format of the **COMPRESS** command is:

```
COMPRESS input_file_specification output_file_specification [options]  
input_file_specification.ext
```

The first parameter specifies the input file. If no path is specified then the current directory is searched for in the input file. The input file name is the only required parameter. By default, COMPRESS.EXE stores the original input file name and extension in the compressed files. This information is used to name the files when they are decompressed.

**output\_file\_specification**

The second parameter specifies the path, and optionally a file name, for the compressed output file. If no output path is specified, then the compressed file is written to the current directory. If no output filename.ext is specified, then the output file retains the name of the input file. If an output filename.ext is specified then it is used to name the compressed output file. The output file replaces any files in the output directory that have the same name except when the **APPEND** option is specified.

See “Compress Options” on page 3-21 for specific options available on the VMS platform.

To supply the secret key and initialization vector in a file, use a hexadecimal editor to create a file and enter their values. VMS users must define the logical name SYS\$COMPRESS to point to the drive and directory where **COMPRESS** and **DECOMP** look for the file named **ENCRYPT.KEY**. See “Encrypting Data with TDCompress” on page 2-18.

## Decompressing on VMS

The format of the **DECOMP** command is:

```
DECOMP input_file_specification output_file.ext [options]
```

### **input\_file**

The first parameter specifies the input file. If no path is specified then the input file is searched for within the current directory. The input filename.ext is the only required parameter.

### **output\_file**

The second parameter specifies the path, and optionally a file name, for the decompressed output file(s). If no output path is specified, then the decompressed files are written to the current directory. Output file names and extensions are obtained from either the directory information embedded in the compressed input files or from the filename.ext specified on the command line. The embedded file names and extensions are used to name the output files if such information exists in the compressed input. Note that the embedded information is used even if the filename.ext parameter is supplied on the command line, unless the **NOINFO** option is specified. The **NOINFO** option causes DECOMP.EXE to ignore any embedded file name and extension information.

If no embedded directory information is contained in the compressed input files, or if the **NOINFO** option is specified, then the filename.ext command-line parameter is used to name the output files. If no output filename.ext is specified, then the output file retains the name of the input file. In any case, the output files will replace any files in the output directory that have the same name except when the **APPEND** option is also specified.

DECOMP.EXE looks for a compression signature in each input file. If an input file does not contain a valid compression signature then DECOMP.EXE simply bypasses the file and continues processing with the next input file. See “**Error! Reference source not found.**” **Error! Bookmark not defined.** for specific options available for the VMS platform.

## VMS Examples

Below are some examples of commands to compress data on VMS:

```
compress compwork.txt [.compout] ascii crlf
```

This command compresses COMPWORK.TXT in the current directory. The input file is compressed into the COMPOUT directory and the output file retains the original input file name. The VMS directory information is embedded in the compressed file and may be used to name the decompressed file.

```
compress data.txt data.cmp ascii crlf append noinfo
```

This example appends DATA.TXT to DATA.CMP, a single output file from multiple input files (the **APPEND** option has been added), and the **NOINFO** option has also been specified. This causes COMPRESS.EXE to omit directory information in the compressed output.

```
compress data.txt data.cmp ascii crlf delete
```

This command compresses DATA.TXT to DATA.CMP in the current directory and will contain the original input file name. The original input file will be deleted after it is compressed.

Below is an example of decompressing data on VMS:

```
decomp comp.cmp delete
```

This command decompresses COMP.CMP in the current directory. The decompressed output files are written to the current directory. The output file names are obtained from the embedded directory information that was supplied by default COMPRESS.EXE processing. The input file COMP.CMP is deleted after the decompression.

Creating archives for like files:

```
compress [.mixed]*.exe exe.compressed savemode  
compress [.mixed]*.com com.compressed savemode crlf ascii
```

Creating archives containing different file types:

```
compress [.mixed]*.exe mixed_appended.compressed savemode - recfm=v lrecl=32767  
compress [.mixed]*.com mixed_appended.compressed append - savemode crlf ascii
```

## Chapter 8: Windows 95/98/NT Platforms

### Installation

#### Diskette or CD

1. Insert the TDCompress disk or CD into the appropriate drive.
2. From a DOS Command window or the Explorer create a directory to store the TDCompress software. Copy the files from the diskette into the new directory.

For example:

```
md c:\commprss
copy a:\*.* c:\commprss
```

3. Run the self-extracting .exe file to decompress the TDCompress executables, libraries and sample programs.

### Installing the TDManager Run-time Files

#### Using IMPORT on Windows 95/98/NT

To complete the security configuration, install the run-time files generated by the TDManager. The run-time files are distributed as a single, compressed and encrypted file. The **IMPORT** utility installs the run-time files. Passphrase files are created with the paths and file names that were entered in TDManager.

1. **cd** to the TDCompress directory  

```
cd \comm-press
```
2. Type **import** and press **Enter** to begin the run-time installation.

**IMPORT** prompts for the name of the compressed run-time file received from the TDManager.

3. Type the name and path of the file and press **Enter**.

**IMPORT** prompts for the name of the directory where the run-time files are to be installed.

- It is recommended to install the run-time files in the TDCompress directory, however, they can be placed in another directory. They must either be in the current directory when the TDCompress programs are run, or the **RUNTIMEPATH** command-line option or environment variable must be used to give the name of the directory.

4. Type the directory name and press **Enter**.

**IMPORT** prompts for the approval code — a 16-character value provided by the TDManager that protects the run-time files from unauthorized access. If you do not know your approval code, then contact the TDManager Administrator.

5. Type the 16-character approval code and press **Enter**.

**IMPORT** prompts for the directory where the **GENKEYS** utility created the **private.key** file.

- Respond with the directory name.
- If an RSA keypair was not generated with the **GENKEYS** utility then press **Enter**.

6. The four prompts described previously can be avoided by providing the following information on the command line:

```
import runtime.rtm \cpdir key=0123456789ABCDEF privkey=\cpdir
```

Installation of the run-time files completes the security configuration. See “Securing Data Using TDCompress” on page 2-12 for details on securing files using TDCompress.

## PC Operation

TDCompress contains two PC programs — COMPRESS.EXE and DECOMP.EXE — which perform data compression and decompression on an IBM PC, or compatible, running Windows 95/98 or NT, DOS version 6.0 or greater, or OS/2 version 3.0 or greater.

The COMPRESS.EXE and DECOMP.EXE programs may be copied to and executed from a directory on a hard drive, or they may be executed from diskette. Fully qualified file names are supported. For example, paths (subdirectories) may be included when specifying file names, and the DOS wildcard characters may be used to specify input and output file masks. In Windows, the programs are executed from an MS-DOS window, **Run** dialog box, or by creating shortcuts.

When multiple input files are compressed on the PC, the user has the option of creating individual compressed output files or appending the compressed files into a single output file (the files will be separated again when decompressed). Default COMPRESS.EXE processing stores the file names and extensions of the input files in the compressed files. This information is used to name the files when they are decompressed. The **DIRNAME** option may be used to store the full paths of the input files along with the filenames and extensions. If directory information is not needed, then the **NOINFO** option may be used to prevent any information from being stored in the compressed files.

By default, COMPRESS.EXE only compresss files that are in the subdirectory specified on the command line (or in the current directory if a full path is not given). The **RECURSE** option causes COMPRESS.EXE to compress matching input files in all subdirectories below the starting directory. The **DIRNAME** option is automatically invoked when the **RECURSE** option is specified.

If the data is delimited by carriage return/line feed pairs, then certain input records can be left uncompressed in the output. This is done by using the **PF=fn** option during compression. The file name (fn) is the name of a parameter file used by COMPRESS.EXE to identify input records that are to be left uncompressed. Further information on using the **PF=fn** option is given in "Compress Options" on page 3-21.

EDI-formatted data can be compressed by specifying the **EDI** option during compression. This option causes COMPRESS.EXE to leave the header and trailer records that mark the beginning and end of an EDI envelope uncompressed. COMPRESS.EXE also ensures that no characters exist in the compressed data that could be misinterpreted as EDI control characters. Further information on using the **EDI** option is given in "Compress Options" on page 3-21.

DECOMP.EXE allows multiple compressed files or segments to exist in the input file. If file name and extension information is available in the compressed segments, then the segments are separated into individually decompressed output files using the name and extension information embedded in the compressed segment. This default processing may be overridden by the user via DECOMP.EXE options.

## Compressing on the PC (DOS, OS/2 and Windows)

The format of the **COMPRESS** command is:

```
COMPRESS [d:\path\]infile.ext [d:\path[\outfile.ext]] [options]
```

[d:\path\]infile.ext

The first parameter specifies the input file(s). Multiple input files may be compressed during a single execution by using the wildcard characters to create a file mask. If no path is specified then the current directory is searched for the input files. The input infile.ext is the only required parameter.

[d:\path[\outfile.ext]]

The second parameter specifies the path and optionally an outfile.ext, for the compressed output file(s). If no output path is specified, then the compressed files is written to the current directory.

If no output filename.ext is specified, then the output files retain the names of the input files. If an output filename.ext is specified then it is used to name the compressed output file. Wildcard characters may be used to create a file mask to uniquely name the output files. The output files replace any files in the output directory that have the same name except when the **APPEND** option is specified.

By default, COMPRESS.EXE stores the original input file name and extension in the compressed files. This information is used to name the files when they are decompressed.

See “Compress Options” on page 3-21 for specific options available for the PC.

## Decompressing on the PC (DOS, OS/2 and Windows)

The format of the **DECOMP** command is:

```
DECOMP [d:\path\]infile.ext [d:\path[\outfile.ext]] [options]
```

```
[d:\path\]infile.ext
```

The first parameter specifies the input file(s). Multiple input files may be decompressed during a single execution by using the wildcard characters to create a file mask. If no path is specified then the current directory is searched for the input files. The input filename.ext is the only required parameter.

```
[d:\path[\outfile.ext]]
```

The second parameter specifies the path and optionally a file name, for the decompressed output file(s). If no output path is specified, then the decompressed files are written to the current directory.

Output file names and extensions are obtained from either the directory information embedded in the compressed input files or from the filename.ext specified on the command line. The embedded file names and extensions are used to name the output files if such information exists in the compressed input. Filename.ext overrides the embedded file name. The **NOINFO** option causes DECOMP.EXE to ignore any embedded file name and extension information.

```
[options]
```

If the compressed input files contain full paths (for example, including the original input subdirectory names), then the **DIRNAME** option may be used to decompress the files into the same subdirectories. The path names stored in the compressed files are appended to the output path name specified on the command line, or to the current directory, to generate the actual path names for the decompressed output files. If a path does not exist, then it is automatically created.

If no embedded directory information is contained in the compressed input files, or if the **NOINFO** option is specified, then the filename.ext command-line parameter is used to name the output files. Wildcard characters may be used to create a file mask to uniquely name the output files. If no output filename.ext is specified, then the output files retain the names of the input files. In any case, the output files replace those in the output directory that have the same name except when the **APPEND** option is specified.

DECOMP.EXE looks for a compression signature in each input file. If an input file does not contain a valid compression signature then DECOMP.EXE simply bypasses the file and continues processing with the next input file. If the **UNCOMP** parm is used, then DECOMP copies the uncompressed file to the output directory.

See “Decompress Options” **Error! Reference source not found.** **Error! Bookmark not defined.** for specific options available for the PC.

## PC Compression/Decompression Examples

Below are some examples of commands to compress/decompress data on the PC (assume a directory named COMPWORK exists on the current drive):

```
compress . \*\compwork ascii crlf
```

This command compresses all files in the current directory. Each input file is compressed into a separate output file in the COMPWORK directory and the output files retain the original input file names. ASCII-to-EBCDIC translation takes place when the files are decompressed on the mainframe or AS/400. Carriage return/line feed sequences are used to delimit the input records and are removed when the files are decompressed on the mainframe. The PC directory information is embedded in the compressed files.

```
compress . \*\compwork\*.cmp ascii crlf
```

This command causes the same processing as the previous example except this command specifies an output file mask. In this example the output files retain the input file names, but carry a .CMP extension.

```
compress . \*\compwork\data.cmp ascii crlf
```

In this example multiple input files are specified, but only one output file, DATA.CMP, is created. An implicit append operation takes place so that DATA.CMP contains all the compressed input files. Note that DATA.CMP is replaced if it exists prior to **COMPRESS** processing (use **APPEND** if DATA.CMP is not to be replaced).

```
compress . \*\compwork\data.cmp ascii crlf append noinfo
```

This example creates a single output file from multiple input files (the **APPEND** option has been added), using the **NOINFO** option. Consequently, COMPRESS.EXE does not include directory information in the compressed output, so that the compressed files cannot be separated into multiple output files when they are decompressed.

```
compress a:\source\*.txt ascii crlf delete
```

This command compresses all the files that are in the SOURCE directory on drive **A:** and have a .txt extension. The files are compressed into the current directory and retain the original input file names. The original input file is deleted after compression.

Below are some examples of decompressing data on the PC.

```
decomp . \*\compwork delete
```

This command decompresses all files in the current directory and writes the decompressed output file to the **COMPWORK** directory. The output file names are obtained from the embedded directory information that was supplied by default **COMPRESS** processing. If no embedded directory information exists, then the output files retain the names of the compressed input files.

```
decomp . \*\compwork noinfo
```

This command again decompresses all files in the current directory. The **NOINFO** options causes **DECOMP** to ignore any embedded directory information. Therefore, the output files retain the names of the compressed input files.

```
decomp . \*\compwork noinfo delete
```

This command adds the **DELETE** function to the previous example. Consequently, the input files are deleted after they are decompressed.

## Compressing PC/Workstation Files

Compression on the PC operates much like the DOS **COPY** command as far as file handling is concerned. Input and output PC file names are specified on the command line. DOS wildcard characters may be used to mask the input and output file names. Multiple input files can be compressed into a single output file or as individual output files. By default, the file name and extension of each input PC file are stored in the compressed file. This information is used to name the files when they are decompressed at a workstation. If the compressed PC data is decompressed on the mainframe, then the mainframe decompression program can separate the compressed PC files into individual sequential files or PDS members. The dynamically allocated sequential files containing the decompressed data are named using stored PC file names and extensions; whereas its decompressed PDS members use a PC file name without the extension. Decompression using dynamic allocation and output PDS processing is specified by execution parameters in the JCL.

## Using GENKEYS On Windows 95/98/NT

The **GENKEYS** utility reads the `easyacc.ini` configuration file from TDManager and creates two output files — **CERTREQ** and **private.key**. Follow the steps below to generate the RSA encryption keys:

1. **cd** to the TDCompress directory
 

```
cd /usr/comm-press
```
2. Run the **GENKEYS** utility.
  - **GENKEYS** prompts for random input data.

3. Type several lines of random characters to ensure the keys are hard to break.
4. Press **Enter** on a blank line to complete the random entry.
  - **GENKEYS** writes the **private.key** and **CERTREQ** files in the TDCompress directory.

The **private.key** file created by **GENKEYS** is permanent and must be retained. The **CERTREQ** file contains the portion of the key that must be certified by the TDManager. The TDManager Administrator issues the security run-time files required to transmit secure data. See “Installing the TDManager Run-time Files” on page 8-63 for a description of the run-time files and details on their installation.

The syntax of the **GENKEYS** command on the workstation is:

```
genkeys path=\path passloc=platform.specific.filename <random.data.file
```

## Input to GENKEYS

The **path=** parameter points to the directory containing the TDAccess configuration file created by TDManager. This file must be named **easyacc.ini**. The configuration file specifies the user's X.500 distinguished name as well as the modulus size and expiration interval to use when generating the RSA keys. **GENKEYS** writes its output files to the same path.

The optional **passloc=** parameter specifies the name of the passphrase location file. This file is where the **IMPORT** job stores random pass-phrases used to encrypt the private key. (See Installing the TDManager Run-time Files” on page 8-63.) The use of the **passloc=** parameter, although optional, is intended to provide an additional point of security for the private key. If no passphrase location is provided, then the passphrase is stored with the private key. The last parameter uses the **STDIN** redirection symbol to provide random data from a file. This data is hashed and used to seed the pseudo-random number generator of **GENKEYS**. It is recommended that some unpredictable data be provided each time that **GENKEYS** is run. If this parameter is omitted, then **GENKEYS** waits for **Enter** to be pressed at the workstation before continuing.

## Output from GENKEYS

The private key generated by **GENKEYS** is written to a file named **privarch.fil** in a security directory under the directory specified by the **path=** parameter (**path/security**). The private key is appended to a previously generated private key. It is important that the **privarch.fil** file is retained for at least as long as the RSA key pair expiration interval. The **privarch.fil** file is used later by the **IMPORT** job to install the security run-time files from the TDManager. (See "Installing the TDManager Run-time Files" on page 8-63.)

The public key generated by **GENKEYS** is written to a file named **certreq.fil** in the same directory where the **privarch.fil** is written. The public key is stored as a PEM-formatted certificate request and must be certified by the TDManager before it can be used. Typically, **certreq.fil** is sent via email or FTP to the TDManager where it is imported and certified. The certificate is returned as part of the security run-time files that are created by TDManager and imported by the **IMPORT** job. (See "Installing the TDManager Run-time Files" on page 8-63.) Although **GENKEYS** appends the public key to any previously generated keys, there is no need to retain **certreq.fil** once the TDManager has imported and certified the request. Informational and error messages from **GENKEYS** are written to **STDOUT**.

## Securing Formatted EDI Data

To secure formatted EDI data, both the EDI and either the **SECURE** or **SECUREONLY** run-time options must be specified during compression. The run-time files created by TDManager as well as any required passphrase files, must be available.

When TDCompress processes the formatted data, the **SENDER** and **RECEIVER** fields from the ISA or UNB segment, along with the group ID from the GS segment, or the transaction type from the ST or UNH segment, are used to perform a search in the lookup table. If a record exists in the lookup table with matching **SENDER**, **RECEIVER**, and **TRANSACTION** values, then the security options specified on the record are used to secure the data in the EDI envelope.

## PC Example for Securing Formatted EDI Data

The command below is an example of how to compress and secure a file (X12EDI.FIL) containing formatted EDI data (X12-format). It will be compressed/secured into a file named X12EDI.CMP in the COMPWORK directory. The run-time files must be in the current directory, unless the **RUNTIMEPATH** environment variable has been set. The compress.log file is written to the current directory, unless the **LOGPATH** environment variable has been set.

```
Compress x12edi.fil \compwork\x12edi.cmp edi secureonly
```

## Unsecuring Formatted EDI Data

To unsecure formatted EDI data, the **EDI** option must be specified during decompression. The TManager run-time files as well as any required passphrase files must be available.

When TDCompress decompresses the secured EDI data, it processes security segments to recover the random bulk encryption key. It then decrypts and decompresses the data and verifies digital signatures. Error messages are created if any security features fail.

### PC Example using ARCHIVE with Secure Formatted EDI Data

The following command is an example of how to decompress and archive a file containing EDIFACT EDI data. In the example, the file EDIFACT.CMP contains authenticated EDIFACT formatted EDI data. It will be decompressed into a file named EDIFACT.DCM in the DCMPWORK directory. The run-time files must be in the current directory, unless the **RUNTIMEPATH** environment variable has been set. The decomp.log file is written to the current directory, unless the **LOGPATH** environment variable has been set. Note that the decomp.log is copied and appended into the archive.log file to be available for future use.

```
decomp edifact.cmp \dcmpwork\edifact.dcm edi archive=c:\save
rename c:\save\archive.log c:\save\archiveold.log
copy c:\save\archiveold.log+c:\compress\compress.log c:\save\archive.log
```

When using the **ARCHIVE** parameter, it is the responsibility of the CA (TManager) administrator to retain certificate information for affected participants. For this reason, the administrator may revoke the certificates for these participants, but may not delete the participant (deleting a participant removes expired and revoked certificates, which are required for the authentication process).

### PC Example for Unsecuring Formatted EDI Data

The following command is an example of how to decompress a file containing X12 EDI data. In the example, the file X12EDI.CMP contains compressed/secured X12-formatted EDI data. It is decompressed into a file named X12EDI.DCM in the DCMPWORK directory. The run-time files must be in the current directory, unless the **RUNTIMEPATH** environment variable has been set. The decomp.log file is written to the current directory, unless the **LOGPATH** environment variable has been set.

```
decomp x12edi.cmp \dcmpwork\x12edi.dcm edi
```

## Securing Non-EDI Data

To secure non-EDI data, the **SECURE** or **SECUREONLY** option must be specified during compression. In addition, the **SECFILE** or **SENDER/RECEIVER/TRANSID** options must be specified during compression. The run-time files created by TDManager as well as any required passphrase files must be available.

Non-EDI data does not contain an independently defined standard header, like the ISA segment present in X12 data, for **COMPRESS** to get **SENDER**, **RECEIVER** and **TRANSID** values. These values can be provided by the user via the **SENDER**, **RECEIVER** and **TRANSID** command-line options or keywords in the **SECFILE**.

If the non-EDI data contains user-defined header records containing the **SENDER**, **RECEIVER** and **TRANSID** values, **SECFILE** keywords can be used to describe the proprietary headers to **COMPRESS**. Details for coding the **SECFILE** are given on page 5-39.

## Unsecuring Non-EDI Data

To unsecure non-EDI data, special run-time options do not need to be specified during decompression. However, the run-time files created by TDManager, as well as any required passphrase files, must be available.

When TDCompress decompresses the secured non-EDI data, it processes the S1S segments to recover the random bulk encryption key, then decrypts and decompresses the data. Digital signatures are verified and error messages are generated if security features fail.

## PC Example for Unsecuring Non-EDI Data

The following command is an example of how to decompress a file containing non-EDI data. In the example, the file DATA.CMP contains compressed/secured non-EDI data. It will be decompressed into a file named DATA.DCM in the DCMPWORK directory. The run-time files must be in the current directory, unless the **RUNTIMEPATH** environment variable has been set. The 'decomp.log' file is written to the current directory, unless the **LOGPATH** environment variable has been set.

```
DECOMP DATA.CMP \DCMPWORK\DATA.DCM EDI
```

## Compressing EDI-Formatted Data

The compression programs on all supported platforms provide a run-time option that can be used to compress EDI-formatted data. X12, UN/ED, UCS EDI data, and EDIFACT formats are supported. The **EDI** option automatically leaves the header and trailer segments uncompressed in the output file. This information must not be compressed so the network can deliver the data correctly. Compressed EDI data can be sent and received using the appropriate **SEND/RECEIVE EDI** commands of the user's communication software.

The above scenarios show only a few of the ways TDCompress can be used to compress data. Other options enable TDCompress to adapt to almost any environment.

# Chapter 9: UNIX/AIX Platform

## Installation

### Diskette/Tape

The `.sfx` file is the self-extracting, TDCompress file in UNIX format.

1. Create a directory to store the TDCompress software. For example:

```
mkdir /usr/commpress  
cd /usr/commpress
```

2. Copy the file into the directory on the UNIX system via one of the following methods:

- Tar from UNIX-formatted diskette or tape.
- FTP from e-mail or Windows-formatted CD.

3. Ensure the user has read/write/execute authority.

4. Change the mode on the self-extracting `.sfx` file so that it can be executed:

```
chmod +X *.sfx
```

5. Execute the file to expand it into its product files — executables, libraries and sample programs.

# Installing the TDManager Run-time Files

## Using IMPORT on UNIX

To complete the security configuration, install the run-time files generated by the TDManager. The run-time files are distributed as a single, compressed and encrypted file. The **IMPORT** utility installs the run-time files. Passphrase files are created with the paths and file names that were entered in TDManager.

1. **cd** to the TDCompress directory.

```
cd /comm-press
```

2. Type **import** and press **Enter** to begin the run-time installation.

**IMPORT** prompts for the name of the compressed run-time file received from the TDManager.

3. Type the name and path of the file and press **Enter**.

**IMPORT** prompts for the name of the directory where the run-time files are to be installed.

- It is recommended to install the run-time files in the TDCompress directory; however, they can be placed in another directory. They must either be in the current directory when the TDCompress programs are run, or the **RUNTIMEPATH** command-line option or environment variable must be used to give the name of the directory.

4. Type the directory name and press **Enter**.

**IMPORT** prompts for the approval code — a 16-character value provided by the TDManager that protects the run-time files from unauthorized access. If you do not know your approval code, then contact the TDManager Administrator.

5. Type the 16-character approval code and press **Enter**.

**IMPORT** prompts for the directory where the **GENKEYS** utility created the private.key file.

- Respond with the directory name.
- If an RSA keypair was not generated with the **GENKEYS** utility then press **Enter**.

6. The four prompts described previously can be avoided by providing the following information on the command line:

```
import runtime.rtm /cpdir key=0123456789ABCDEF privkey=/cpdir
```

Installation of the run-time files completes the security configuration. See “Securing Data Using TDCompress” on page 2-12 for details on securing files using TDCompress.

## UNIX Operation

TDCompress contains two UNIX programs — COMPX and DECOMPX — that perform data compression and decompression on UNIX operating systems.

The programs read a list of fully qualified file names from standard input and compress/decompress the files as specified by other command-line options. When multiple input files are compressed, the user has the option of creating individual compressed output files or appending them to a single file (the files are separated when decompressed). Default COMPX processing stores the file name with the compressed output files. This information is used to name the files when they are decompressed. The **DIRNAME** option may be used to store the full paths of the input files (for example, all subdirectories) along with the file names. If directory information is not needed, then the **NOINFO** option may be used to prevent any information from being stored in the compressed files.

If the data is delimited by line feeds, then certain input records can be left uncompressed in the output. This is done by using the **PF=fn** option during compression. The file name is the name of a parameter file used by COMPX to identify input records that are to be left uncompressed. Further details are given in “Compressing on UNIX/AIX” on page 9-77.

EDI data can be compressed by specifying the **EDI** option during compression. This option causes COMPX to automatically leave the header and trailer records that mark the beginning and end of an EDI envelope uncompressed. COMPX also ensures that no characters exist in the compressed data that could be misinterpreted as EDI control characters.

DECOMPX allows multiple compressed files or segments to exist in the input file. If the file names are available in the compressed segments, then the segments are separated into individually decompressed output files using the name embedded in the compressed segment. This default processing may be overridden by the user via DECOMPX options.

## Compressing on UNIX/AIX

The format of the COMPX command is:

```
compx [ [/path/]outfile ] [options]
```

COMPX reads a list of fully qualified file names from standard input. Each input file in the list is compressed according to the command-line options specified and written to the output file. The list of input files can be generated by a Find command that is piped to COMPX, or the user may use an editor to create a file containing the list and then invoke COMPX redirecting **STDIN** to the file.

COMPX parameters may be specified in any order. All options are reserved, keyword options that may not be used as the output file name unless a full path is specified. The first parameter that is not recognized as a valid option is interpreted by COMPX to be the path and optional file name, for the compressed output file(s). If an output path is not specified, then the compressed files are written to the current directory. If a name is not specified, then the output files retain the names of the input files. If an output file name is specified then it is used to name the compressed output file, and all the input files are compressed into a single output file, then separated when decompressed. The output files replace any files in the output directory that have the same name except when the **APPEND** option is specified.

By default, COMPX stores the original input file name in the compressed files. This information is used to name the files when they are decompressed.

See “Compress Options” on page 3-21 for specific options available for the UNIX platforms.

## Decompressing on UNIX/AIX

The format of the DECOMPX command is:

```
decompx [ /path/[outfile] ] [options]
```

DECOMPX reads a list of fully qualified file names from standard input. Each input file in the list is decompressed according to the command-line options specified and written to the output file. The list of input files can be generated by a FIND command that is piped to DECOMPX, or the user may use an editor to create a file containing the list and then invoke DECOMPX redirecting **STDIN** to the file.

DECOMPX parameters may be specified in any order. All options are reserved, keyword options and may not be used as the output file name unless a full path is specified. The first parameter that is not recognized as a valid option is interpreted by DECOMPX to be the path and optional file name, for the decompressed output file(s). If no output path is specified, then the decompressed files are written to the current directory.

Output file names are obtained from either the directory information embedded in the compressed input files or from the file name specified on the command line. The embedded file names are used to name the output files if such information exists in the compressed input. Note that the embedded information is used even if the file name parameter is supplied on the command line, unless the **NOINFO** option is also specified. The **NOINFO** option causes DECOMPX to ignore any embedded file name and extension information.

If the compressed input files contain full paths (for example, including the original input subdirectory names), then the **DIRNAME** option may be used to decompress the files into the same subdirectories. The paths stored in the compressed files are appended to the output path specified on the command line, or to the current directory, to generate the actual paths for the decompressed output files. If a path does not exist, then it is automatically created.

If no embedded directory information is contained in the compressed input files, or if the **NOINFO** option is specified, then the file name command-line parameter is used to name the output file. If an output file name is not specified, then the output files retain the names of the input files. In any case, the output files will replace any files in the output directory that have the same name except when the **APPEND** option is also specified.

DECOMPX looks for a compression signature in each input file. If an input file does not contain a valid compression signature then DECOMPX simply bypasses the file and continues processing with the next input file. If the **UNCOMP** parm is used, then DECOMPX copies the uncompressed file to the output directory.

## UNIX/AIX Examples

Below are some examples of commands to compress data under UNIX and AIX (assume a directory named /compwork exists on the current drive):

```
find ./ * -name '*' -print | compx /compwork ascii crlf
```

This command compresses all files in the current directory. Each input file is compressed into a separate output file in the /compwork directory and the output files retain the original input file names. ASCII-to-EBCDIC translation takes place when the files are decompressed on the mainframe. Line feeds are used to delimit the input records but are removed when the files are decompressed on the mainframe. The directory information is embedded in the compressed files and may be used to name the decompressed files.

```
find ./ * -name '*' -print | compx /compwork/data.cmp ascii crlf
```

In this example, multiple input files are specified, but only one output file — data.cmp — is created. An implicit append operation takes place so that data.cmp will contain all the compressed input files. Note that data.cmp is replaced if it already exists prior to COMPX processing (the **APPEND** option should be used if data.cmp is not to be replaced).

```
find ./ * -name '*' -print | compx /compwork/data.cmp ascii crlf append noinfo
```

This example creates a single output file from multiple input files—by adding the **APPEND** option. It does not contain directory information in the compressed output from COMPX as specified by the **NOINFO** option.

```
find /source -name '*.txt' -print | compx ascii crlf delete
```

This command compresses all the files in the /source directory that end with .txt and retains their original input file names. The original input files are deleted after they are compressed.

```
ls -l /home/datain/*.txt | compx /compwork ascii crlf
```

This example uses the `ls` command instead of `Find`. Note that the input path must be fully qualified, even though the files are in the current directory.

Below are examples of decompressing data on UNIX/AIX:

```
find ./ * -name '*' -print | decompx /dcmpwork delete
```

This command decompresses all files in the current directory and writes them to the `/dcmpwork` directory. The output file names are obtained from the embedded directory information supplied by default `COMPX` processing. If no embedded directory information exists, then the output files retain the names of the compressed input files.

```
find ./ * -name '*' -print | decompx /dcmpwork noinfo
```

This command again decompresses all files in the current directory. The `NOINFO` option causes `DECOMPX` to ignore any embedded directory information. Therefore, the output files retain the names of the compressed input files.

```
find ./ * -name '*' -print | decompx /dcmpwork noinfo delete
```

This command adds the Delete function to the previous example, which deletes the input files after they are decompressed.

```
ls -l home/compwork/*.cmp | decompx /dcmpwork
```

This example uses the `ls` command instead of the `Find`. Note that the input file path must be fully qualified, even though the files are in the current directory.

## Using GENKEYS on UNIX

The `GENKEYS` utility reads the `easyacc.ini` configuration file from `TDManager` and creates two output files — `CERTREQ` and `private.key`. Follow the steps below to generate the RSA encryption keys:

1. `cd` to the `TDCompress` directory.  

```
cd /usr/comm-press
```
2. Run the `GENKEYS` utility.
  - `GENKEYS` prompts for random input data.
3. Type several lines of random characters to ensure the keys are hard to break.
4. Press **Enter** on a blank line to complete the random entry.
  - `GENKEYS` writes the `private.key` and `CERTREQ` files in the `TDCompress` directory.

The **private.key** file created by **GENKEYS** is a permanent key file and must be retained. The **CERTREQ** file contains the portion of the key that must be certified by the TDManager. The TDManager Administrator issues the security run-time files required to transmit secure data. See “Installing the TDManager Run-time Files” on page 9-75 for a description of the run-time files and details on their installation.

The syntax of the **GENKEYS** command on the workstation is:

```
genkeys path=/path passloc=platform.specific.filename <random.data.file
```

## Input to GENKEYS

The **path=** parameter points to the directory containing the TDAccess configuration file created by TDManager. This file must be named **easyacc.ini**. The configuration file specifies the user's X.500 distinguished name as well as the modulus size and expiration interval to use when generating the RSA keys. **GENKEYS** writes its output files to the same path.

The optional **passloc=** parameter specifies the name of the passphrase location file, which is where the **IMPORT** job stores the random passphrase used to encrypt the private key. (See "Installing the TDManager Run-time Files" on page 9-75.) Use of the parameter, although optional, is intended to provide an additional point of security for the private key. If a passphrase location is not provided, then the passphrase is stored with the private key. The last parameter uses the **STDIN** redirection symbol to provide random data from a file. This data is hashed and used to seed the pseudo-random number generator of **GENKEYS**. It is recommended that some unpredictable data be provided each time that **GENKEYS** is run. If **STDIN** redirection is not used, then **GENKEYS** waits for random data input from the keyboard. Pressing **Enter** on a blank line ends the input and **GENKEYS** continues.

## Output from GENKEYS

The private key generated by **GENKEYS** is written to a file named **privarch.fil** in a /security directory under the directory specified by the **path=** parameter (**path/security**). The private key is appended to a previously generated private key. It is important that the **privarch.fil** file is retained for at least as long as the RSA key pair expiration interval. The **privarch.fil** file is used later by the **IMPORT** job to install the security run-time files from the TDManager. ("Installing the TDManager Run-time Files" on page 9-75.)

The public key generated by **GENKEYS** is written to a file named **certreq.fil** in the same directory where the **privarch.fil** was written. The public key is stored as a PEM-formatted certificate request and must be certified by the TDManager before it can be used. Typically, **certreq.fil** is sent via email or FTP to the TDManager where it is imported and certified. The certificate is returned as part of the security run-time files that are created by TDManager and imported by the **IMPORT** job. (See "Installing the TDManager Run-time Files" on page 9-75.) Although **GENKEYS** appends the public key to any previously generated keys, there is no need to retain **certreq.fil** once the TDManager has imported and certified the request. Informational and error messages from **GENKEYS** are written to **STDOUT**.

# Chapter 10: OS/400 Platform

## Installation

### Diskette

Installation of TDCompress from diskette requires TCP/IP and the FTP server to be installed and configured on OS/400. The TDCompress software is distributed as a saved library in SAVEFILE format. Do the following to install the software:

1. Create an empty save file on the AS/400. For example: CRTSAVF SAVEFILE
2. Upload the TDCompress save file into the new AS/400 save file using binary mode FTP. For example:

<code>ftp 2.72.125.43</code>	<b>&lt;= connect to AS/400</b>
<code>220 User (as/400:(none)): userid</code>	<b>&lt;= enter USERID</b>
<code>331 Enter password.</code>	<b>&lt;= enter PASSWORD</b>
<code>230 USERID logged on.</code>	
<code>ftp&gt; bin</code>	<b>&lt;= binary mode</b>
<code>220 Representation type is binary IMAGE.</code>	
<code>ftp&gt; put cplib.savf SAVEFILE</code>	<b>&lt;= transfer save file</b>
<code>200 PORT subcommand request successful.</code>	
<code>150 Sending file to member SAVEFILE in file SAVEFILE.</code>	
<code>250 File transfer completed successfully.</code>	
<code>ftp&gt; quit</code>	<b>&lt;= disconnect</b>

3. Restore the TDCompress library from the save file. For example:

```
RSTLIB SAVLIB(CPLIB) DEV(*SAVF) SAVF(SAVEFILE)
```

### Tape

Type the following command to unload the TDCompress library from the distribution tape:

```
RSTOBJ OBJ(*ALL) SAVLIB(CPLIB) DEV(TAPxx) RSTLIB(xxxxxxxx)
```

Substitute correct values for DEV and RSTLIB. Substitute the library name printed on the tape's external label with the name of the SAVLIB library.

## Installing the TDManager Run-time Files

### Using IMPORT on OS/400

To complete the security configuration, you must install the run-time files generated by the TDManager. The run-time files are distributed as single, compressed and encrypted files. The IMPORT utility installs the run-time files:

1. Make the TDCompress library the current library:

```
CHGCURLIB CPLIB
```

2. Call **IMPORT** to begin the run-time installation:

```
CALL IMPORT
```

- **IMPORT** prompts for the name of the compressed run-time file received from TDManager. If the file cannot be located via the LIBLIST, then enter the name using the LIBRARY/FILENAME format. Otherwise, enter the filename.
- **IMPORT** prompts for the name of the library where the security run-time files are to be installed. Respond with the name of the TDCompress library.
- **IMPORT** prompts for the approval code. This is a 16-character value provided by the TDManager that protects the run-time files from unauthorized access. If you do not know your approval code, then contact the TDManager Administrator.
- **IMPORT** prompts for the library where the **GENKEYS** utility created the privkey file. Respond with the library name. (If you did not generate your RSA keypair with the **GENKEYS** utility, then press **Enter**.)

3. The four prompts described previously can be avoided by providing all the information on the **CALL**. The following is an example:

```
CALL IMPORT PARM('RTMFILE' 'CPLIB' 'KEY=0123456789ABCDEF' 'PRIVKEY=CPLIB441')
```

Installation of the run-time files completes the security configuration. See “Securing Data Using TDCompress” on page 2-12 for details on securing files using TDCompress.

## AS/400 Operation

TDCompress for the AS/400 is distributed as two programs — **COMPRESS** and **DECOMP**. To execute the programs OS/400 V3R7M0 or greater is required.

## Compressing on the AS/400

The **COMPRESS** program is invoked via a CALL CL statement. Options are specified as individually quoted strings in the **PARM** field of the CALL statement. **COMPRESS** assumes default names for the input and output files of **DATAIN** and **DATAOT**, respectively. Example:

```
CALL libname/COMPRESS PARM('ASCII' 'CRLF')
```

The default input and output file names may be overridden using the OVRDBF command. Example:

```
OVRDBF FILE(DATAIN) TOFILE(libname/filein)
OVRDBF FILE(DATAOT) TOFILE(libname/fileout)
CALL libname/COMPRESS PARM('ASCII' 'CRLF')
```

An alternate method for providing input and output file names is to include them as the first two parameters in the **PARM** list:

```
CALL libname/COMPRESS PARM('lib/filein' 'lib/fileout' 'ASCII' 'CRLF')
```

If the output file does not exist, **COMPRESS** creates it using system default values for record length and file size. If the output file does exist, then any data is overwritten unless the **APPEND** option is used.

See “Compress Options” on page 3-21 for specific options available for the AS/400.

## Decompressing on the AS/400

The **DECOMP** program is invoked via a CALL CL statement. Options are specified as individually quoted strings in the **PARM** field of the CALL statement.

**DECOMP** assumes default names for the input and output files of **DATAIN** and **DATAOT**, respectively.

Example:

```
CALL libname/DECOMP PARM('QUIET')
```

The default input and output file names may be overridden using the OVRDBF command. Example:

```
OVRDBF FILE(DATAIN) TOFILE(libname/filein)
OVRDBF FILE(DATAOT) TOFILE(libname/fileout)
CALL libname/DECOMP PARM('QUIET')
```

An alternate method for providing input and output file names is to include them as the first two parameters in the **PARM** list:

```
CALL libname/DECOMP PARM('lib/filein' 'lib/fileout' 'QUIET')
```

If the output file does not exist, **DECOMP** creates it using system default values for record length and file size. If the output file does exist, then any data is overwritten unless the **APPEND** option is used.

See “**Error! Reference source not found.**” **Error! Bookmark not defined.** for specific options available for the AS/400.

## Using GENKEYS on OS/400

1. Make the TDCompress library the current library:
  - **CHGCURLIB CPLIB**
2. Type **CALL GENKEYS** to create the RSA encryption keys.
  - **GENKEYS** prompts for random input data.
3. Type several lines of random characters to ensure the keys are hard to break.
4. Press the **Enter** key on a blank line to complete the random entry.
  - **GENKEYS** writes the **PRIVKEY** and **CERTREQ** files in the TDCompress library.

The **PRIVKEY** file created by **GENKEYS** is a permanent key file and must be retained. The **CERTREQ** file contains the portion of the key that must be certified by the TDManager. The TDManager Administrator issues the security run-time files required to transmit secure data. See “Installing the TDManager Run-time Files” on page 10-83 for a description of using **GENKEYS**.

The **GENKEYS** utility reads the **EASYACC** configuration file from TDManager and creates two output files — **CERTREQ** and **PRIVKEY**. Sample **GENKEYS** JCL is distributed in the **SAMPLIB** PDS. Instructions for running the job and creating your keys are included in the sample JCL.

The **PRIVKEY** file created by **GENKEYS** is a permanent key file and must be retained. The **CERTREQ** file contains the portion of the key that must be certified by the TDManager. The TDManager Administrator issues the security run-time files required to transmit secure data. See “Installing the TDManager Run-time Files” on page 10-83 for a description of the run-time files and details on how to install them.

## Input to GENKEYS

The **INIFILE DD** points to the TDAccess configuration file created by TDManager. This file specifies the user's X.500 distinguished name as well as the modulus size and expiration interval to use when generating the RSA keys. The **SYSIN DD** provides an optional control statement that specifies the name of the passphrase location file. This file is where the **IMPORT** job stores the random passphrase used to encrypt the private key (see "Installing the TDManager Run-time Files" on page 6-44). Its use, although optional, is intended to provide an additional point of security for the private key. If no passphrase location is provided, then the passphrase is stored with the private key. The format of the **PASSLOC** statement is:

**PASSLOC=platform.specific.filename**

Any other input specified via the **SYSIN DD** is hashed and used to seed **GENKEYS'** pseudo-random number generator. It is recommended that some unpredictable data be provided each time that **GENKEYS** is run.

## Output from GENKEYS

The private key generated by **GENKEYS** is written to the **PRIVARCH DD**. The private key is appended to any previously generated private keys. It is important that the **PRIVARCH** file be retained for at least as long as the RSA keypair expiration interval. The **PRIVARCH** file is used later by the **IMPORT** job to install the security run-time files from the TDManager (See "Installing the TDManager Run-time Files" on page 6-44).

The public key generated by **GENKEYS** is written to the **CERTREQ DD**. The public key is stored as a PEM-formatted certificate request and must be certified by the TDManager before it can be used. Typically, the **CERT.REQ** file is sent (perhaps via email or FTP) to the TDManager where it is imported and certified. The certificate is returned as part of the security run-time files that are created by TDManager and imported by the **IMPORT** job. (See "Installing the TDManager Run-time Files" on page 6-44.) Although **GENKEYS** appends the public key to any previously generated keys, there is no need to retain the **CERTREQ** file once the TDManager has imported and certified the request.

Informational and error messages from **GENKEYS** are written to the **SYSPRINT DD**.

## Modifying the Default Translation Tables

TDCompress contains a default EBCDIC-to-ASCII translation table. The user can modify the table at run time if the default translation is not appropriate.

All translation between the EBCDIC and ASCII character sets is performed on the EBCDIC host (for example, MVS or AS/400). Translation from EBCDIC to ASCII occurs during compression when the ASCII execution parameter is specified. Translation from ASCII to EBCDIC occurs during decompression if the ASCII parameter was supplied when the data was compressed.

The default table is replaced by providing a file with the desired EBCDIC-to-ASCII translation. The name of the file is specified via the **TRANTBL=** parameter. The file must contain 256 pairs of characters. Each pair of characters represent one hexadecimal ASCII byte. The position of each pair is the EBCDIC value to translate and the value of each pair is the ASCII value to translate to. For example, the first pair of characters gives the ASCII translation for the EBCDIC value x'00', the second pair gives the ASCII translation for the EBCDIC value x'01' and so on. The last pair gives the ASCII translation for the EBCDIC value x'FF'. Blanks or commas can be used to separate the pairs for readability.

Below is the TDCompress default EBCDIC-to-ASCII translation table as it would be entered in a **TRANTBL** file:

```

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
80 81 82 83 84 8E 87 8F 88 89 8A 8B 8C 8D 86 7F
90 91 97 93 94 95 96 9C 98 99 9A 9B 9D 85 9E 92
20 A0 A1 A2 A3 A4 A5 A6 A7 A8 D5 2E 3C 28 2B 7C
26 A9 AA AB AC AD AE AF B0 B1 21 24 2A 29 3B 5E
2D 2F B2 B3 B4 B5 B6 B7 B8 B9 E5 2C 25 5F 3E 3F
BA BB BC BD BE BF C0 C1 C2 60 3A 23 40 27 3D 22
C3 61 62 63 64 65 66 67 68 69 C4 C5 C6 C7 C8 C9
CA 6A 6B 6C 6D 6E 6F 70 71 72 CB CC CD CE CF D0
D1 7E 73 74 75 76 77 78 79 7A D2 D3 D4 5B D6 D7
D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 5D E6 E7
7B 41 42 43 44 45 46 47 48 49 E8 E9 EA EB EC ED
7D 4A 4B 4C 4D 4E 4F 50 51 52 EE EF F0 F1 F2 F3
5C 9F 53 54 55 56 57 58 59 5A F4 F5 F6 F7 F8 F9
30 31 32 33 34 35 36 37 38 39 FA FB FC FD FE FF

```

# Appendix A

## Using API (Extended Security Option)

Application programmers can call the TDCompress utilities to compress, decompress, and secure data within their applications. A platform-specific static library, DLL, or shared object is provided that contains the TDCompress utilities as function calls. The libraries contain separate functions for compressing and securing data, and for decompressing and unsecuring data. All the features of the standalone utilities are available via the CALL interface.

Data is compressed and secured by calling either the compress function or the compprog function. Data is decompressed and unsecured by calling either the decomp function or the dcmpprog function. The functions differ in the specific interface between them and the caller and in how input and output is handled.

The compress and decomp functions expect to receive a parameter list like the parameter list built by the operating system when using the standalone utilities. The parameter list contains the input and output file names and any keyword options required. The called compress and decomp routines operate like the standalone utilities, reading and writing files, and performing the requested compression and security. The compress and decomp entry points are referred to as the command-line API because they operate as if they had been executed from the command line.

The compprog and dcmpprog functions require the calling program to set up a parameter block used to specify options and pass status information between the caller and the functions. The calling program is responsible for all input and output operations; compprog and dcmpprog do not perform file handling. Instead, the calling program passes and receives data from the functions in memory buffers. The compprog and dcmpprog entry points are referred to as the interactive API because they interact with the calling program to process data.

## Using the Command-Line API (COMPRESS and DECOMP)

When a C program is executed from the command line, the operating system passes two parameters to the program's main function. The first parameter is an integer that is the number of values entered on the command line. The second parameter is the address of an array of pointers to the values entered on the command line. The first value pointed to in the array is the name of the executed program itself, so there is always at least one value present. For example, to compress and secure a file named filein to the output file fileout, the command line might look like this:

```
compress filein fileout ascii crlf secure
```

When the compress program executes, it receives two parameters from the operating system — an integer, traditionally given the variable name `argc`, and the address of an array of pointers, traditionally given the variable name `argv[]`. For the example above, `argc` has a value of 6 and `argv[]` contains six pointers — a pointer to each value entered on the command line.

Calling programs can emulate the operating system by building the `argv[]` array and then calling `compress` or `decomp` with the `argc` and `argv[]` parameters. For a program to emulate the operating system using the command-line example above, it needs to allocate and initialize the six values as string variables. The `argv` array must be initialized to contain six pointers to the string variables. The program calls `compress` passing two parameters — the integer 6 and the address of the `argv` array. The following code fragment demonstrates this:

```
#include "comprss.h"
int CallCompress( )
{
    // declare and initialize local variables
    //
    int    status;
    char   *newArgv[6]      =      {
                                "compress",
                                "filein",
                                "fileout",
                                "ascii",
                                "crlf",
                                "secure"
                                };

    // call compression routine and return completion code
    //
    status = compress( 6, newArgv );
    return status;
}
```

## Using the Interactive API (compprog and dcmpprog)

A program that handles its own files, or that processes data that is not file oriented, calls the `compprog` and `dcmpprog` functions to add or remove compression and security. The caller sets up a parameter block that specifies the desired compression and decompression options. Most of the options correspond to one of the TDCompress command-line options, for example, **EDI**, **SECURE**, etc. Data is passed to and from the TDCompress functions in memory buffers. Next, a conversation takes place between the calling program and the TDCompress functions, with the calling program providing additional input when requested, and TDCompress giving output as needed until all data is processed.

The name of the parameter block structure is CALLPBLK. It is defined in the parmbld.h header file and is not reprinted here. Other definitions, including the TDCompress function prototypes and calling conventions are in the comprss.h and cpapi.h header files. Because sample programs are distributed with the TDCompress software to demonstrate the compprog and dcmpprog functions, only a brief overview is given below.

## The Interactive API Conversation

The calling program must initialize the parameter block to NULLs and then put the size of (CALLPBLK) in the first field of the block. This indicates the version of CALLPBLK to the TDCompress functions.

The caller must allocate input and output buffers and set the maximum length of the output buffer in the **outbufLength** field in CALLPBLK. The maximum buffer size is 4,294,967,295 bytes (4G -1). When securing X12 or EDIFACT EDI data, the output buffer must be large enough to hold an entire, secured EDI envelope (for example, from the ISA through the IEA segments of X12 data and from the UNA/UNB through the UNZ segments of EDIFACT data).

Desired options are requested by setting the various option flags to 1. All TDCompress command-line options are available with the exception of specific file and directory handling options. Certain text options, such as the path where the security run-time files are located, must be placed in the appropriate CALLPBLK string fields.

To begin a conversation, the caller fills the first input buffer and sets its length in the **inbufLength** field in CALLPBLK. The caller then invokes compprog or dcmpprog passing the addresses of the buffers and the address of the parameter block as parameters. Upon return, the caller must check the **rc**, **inrq** and **otrq** fields in CALLPBLK.

If **rc** is non-zero, then an error occurred and corrective action is needed. Use the value in the **rc** field to look up the error in "COMPRESS and DECOMP Error Messages and Codes" on page 10-95.

If **inrq** is non-zero, then compprog or dcmpprog is making an input request for a new input buffer. The caller must fill the input buffer and set its length in the **inbufLength** field in CALLPBLK. Next, the caller must re-invoke compprog or dcmpprog.

When calling compprog on a record-based operating system, such as MVS and OS/400, if the **CRLF** option is specified, then the caller must supply input records one at a time. This allows compprog to correctly delimit multiple records.

If **otrq** is non-zero, then compprog or dcmpprog has filled the output buffer, and is making an output request. The actual length of the output buffer filled by the TDCompress function is set in the **usedOutbufLength** field in CALLPBLK. The caller must dispose of the filled output buffer and re-invoke compprog or dcmpprog.

When calling `dcmpprog` on a record-based operating system, such as MVS and OS/400, if the data is compressed with the **CRLF** option then output records are returned one at a time. This allows the caller to distinguish between multiple records.

This conversation between the calling program and the TDCompress function continues until either the caller wishes to stop compressing/securing input, or until input has been completely decompressed/unsecured.

## Ending the Conversation

The beginning of the end of a conversation with the `compprog` function happens when the caller sets the **eod** field in `CALLPBLK` to a non-zero value. The caller sets **eod** in response to an input request from `compprog`. This is the beginning of the end of the conversation because `compprog` will more than likely need to return at least one output buffer. The conversation is actually ended when `compprog` returns with both the **inrq** field and the **otrq** field set to zero. The caller can begin a new conversation by once again filling the input buffer, setting its length in the **inbufLength** field and invoking `compprog`.

When calling `dcmpprog`, the caller is not responsible for ending the conversation. Under normal circumstances, `dcmpprog` detects the end of compressed data automatically and, after returning the last output buffer, returns with both the **inrq** field and the **otrq** field set to zero. The **inbufLength** field is set to the length of the unprocessed data remaining in the last input buffer.

# Appendix B

## Technical Notes

Conversion between the ASCII and EBCDIC formats is always performed on the EBCDIC machine (for example, the mainframe or AS/400). If the `ASCII` option is specified during compression on the mainframe or AS/400, then the records are translated to ASCII format prior to being compressed. Data compressed on a PC with the `ASCII` option is translated to EBCDI format during decompression on the mainframe or AS/400.

The `CRLF` option of TDCompress causes a `x'1E'` to be used as a record separator byte. When compressing data on the mainframe or AS/400, the record separator is added at the end of each input record before it is compressed. When compressing data on the PC, the record separator byte replaces the carriage return/line feed pair that delimits each record. When the `DECOMP` program on the mainframe or AS/400 encounters a record separator, it removes the `x'1E'` and writes a decompressed record to the output file. Fixed length output records are padded with blanks, if necessary. The `DECOMP` program on the PC replaces the record separators with carriage return/line feed pairs to delimit the decompressed output records. The `CRLF` option must be specified when variable length records are compressed if the decompressed records are to retain their original record lengths.

Compressed files contain a signature in the first 16 bytes of the first compressed record. The format of the signature follows:

Byte Pos	Value	ASCII value and description
1-2	<code>x'EEED'</code>	constant
3-10	<code>c'COMPRESS'</code>	constant
11	<code>x'40'</code>	TDCompress version number
12	flag byte	(first 4 bits complete the TDCompress version number)
	<code>b'00000001'</code>	<code>x'01'</code> directory info is embedded
	<code>b'00000010'</code>	<code>x'02'</code> ASCII specified during compression
	<code>b'00000100'</code>	<code>x'04'</code> CRLF specified during compression

13	b'00000011'	x'03' compressed data encrypted with <b>DES</b>
	b'00000101'	x'05' compressed data encrypted with <b>RC2</b>
	b'00000111'	x'07' compressed data encrypted with triple <b>DES</b>
	b'00010000'	x'10' 21 <sup>st</sup> century bit
14-16	reserved	

If directory information is embedded in the compressed data it begins in position 17 as follows:

Length	Format	Description
4	HHMM	time from directory entry of PC file
6	YYMMDD	date from directory entry of PC file
1	binary	length of filename
var	character	filename

The compressed data immediately follows the signature, which is composed of variable length blocks. Each block is preceded by a three byte block length in packed decimal format. The last two bytes of each block is a 16-bit CRC used to verify data integrity.

# Appendix C

## Error Messages and Codes

The TDCompress programs write informational and error messages to report on their activity. The messages are usually written to the console (or the **STDOUT** file) as well as the log files. Each message has an associated error code, which is printed at the start of the message.

The programs return with the highest error code encountered during processing. The error code can be tested using facilities provided by the operating system, such as the DOS ERRORLEVEL mechanism, UNIX shell programming functions, or MVS condition code testing. User programs calling the TDCompress application programming interface must check the **RC** field in the parameter block after each call to determine if an error occurred.

## COMPRESS and DECOMP Error Messages and Codes

RC=1	<p>Error allocating memory.</p> <p>A failure occurred allocating memory for work areas.</p> <p><b>ACTION:</b> Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun <b>COMPRESS</b> or <b>DECOMP</b>.</p>
RC=9	<p>File xxxxxxxx is not a compressed file.</p> <p>The specified input file does not contain any compressed data and the UNCOMP parameter was not specified.</p> <p>For MVS or AS/400 platforms, this message will only be produced if you specify the NOUNCOMP parameter.</p> <p><b>ACTION:</b> This is an informational message. It is issued as a precaution to the user that an input file was not recognized by DECOMP as a compressed file. On ASCII platforms, use the UNCOMP parameter to suppress this message.</p>

RC=10            File xxxxxxxx is not an EDI file.

The specified input file does not contain EDI data.

ACTION: This is an informational message. It is issued as a precaution to the user that an input file was not recognized by **COMPRESS** or **DECOMP** as an EDI file.

RC=14            Compressed data ended prematurely for input file.

A new compressed segment started, or end-of-file was reached, before the current segment was complete. Processing continues with the next compressed file.

ACTION: The compressed data has been altered, or corrupted, and cannot be accurately decompressed.

RC=15            CRC failure (reason code=15) for input file.

**DECOMP** detected a failure during cyclic redundancy checking. Processing continues with the next compressed file.

The compressed data has been altered, or corrupted, and cannot be accurately decompressed. **DECOMP** bypasses the corrupted file and continues decompressing with the next compressed file.

RC=16            Invalid signature.

ACTION: Contact bTrade.com at 800-425-0444 for further information

RC=17            Unable to decompress data due to restricted license violation.

The user has a restricted license for TDCompress software and can only use it with specific trading partners. Compressed data can only be exchanged and decompressed with those partners.

ACTION: Contact bTrade.com at 800-425-0444 for information on obtaining a full, unrestricted license.

RC=18            Invalid EDI envelope.

The EDI data being compressed or decompressed contains an incomplete envelope. This error can also occur when an invalid segment terminator is encountered.

ACTION: If compressing, do not use carriage return, line feed or new line characters as the segment terminator. Correct the EDI envelope and rerun **COMPRESS**. If decompressing, then the file has most likely been corrupted. Processing continues with the next envelope.

RC=20            Error opening input file.

An error occurred when the specified input file was being opened.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=21            Error reading input file.

An error occurred when the specified input file was being read.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=24            Error positioning input file.

An error occurred when the input file was being positioned for decompression.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.

RC=30            Error opening output file.

An error occurred when the specified output file was being opened.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=31            Error reading output file.

An error occurred when the specified file was being read.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS**.

RC=32            Error writing output file.

An error occurred when the specified output file was being written.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=33            Error closing output file.

An error occurred when the specified output file was being closed.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=34            Error positioning output file.

An error occurred when the output file was being positioned for compression.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS**.

RC=35            Error retrieving info for file.

An error occurred while retrieving the MVS DCB parameters for the specified file.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=36            Disk is full when log file is created. No permissions exist when trying to write to the log file. File is in use by other user.

ACTION: Determine which problem exists and correct it.

RC=37            PDS output not allowed with EDI or APPEND parms

**DATAOT** must be a sequential file when using the EDI or APPEND options.

ACTION: Change the **DATAOT** DD to use a sequential dataset and rerun **COMPRESS** or **DECOMP**.

RC=38            SYSUT1 and DATAXX files cannot be PDS.

SYSUT1 and DATAXX must be sequential files.

ACTION: Change the SYSUT1 and/or DATAXX DD to use a sequential dataset and rerun **DECOMP**.

RC=50            Error opening encrypt.key file.

An error occurred opening the file that contains the encryption/decryption key.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=51            Error reading encrypt.key file.

An error occurred reading the file that contains the encryption/decryption key.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

RC=52            Proprietary encryption not supported in this version.

Either the **ENCRYPT** parameter was specified as a **COMPRESS** or **DECOMP** parm or the file being processed by **DECOMP** was encrypted using the **ENCRYPT** parameter from an older version of the product.

ACTION: Use **RC2**, **DES**, or **DE3** to encrypt the file.

RC=54            Error decrypting file (invalid pad character).

The specified file did not decrypt successfully during **DECOMP** processing.

ACTION: A corrupted compressed file most likely causes this. The file cannot be accurately decompressed. **DECOMP** bypasses the file and continues decompressing with the next compressed file.

RC=55            Error decrypting file.

The specified file did not decrypt successfully during **DECOMP** processing.

ACTION: An incorrect decryption key most likely causes this. The same key used to encrypt the data during compression must be used to decrypt the data during decompression. Provide the correct key in the encrypt.key file and rerun **DECOMP**.

RC=56            This message should not occur, since bTrade.com only builds for triple **TRIPLE DES** encryption.

ACTION: Should this message occur, contact bTrade.com and return the software.

- RC=57      This message should not occur, since bTrade.com only builds for triple **TRIPLE DES** encryption.
- ACTION: Should this message occur, contact bTrade.com and return the software.
- RC=58      Encryption hardware error.
- An error was returned when attempting to access the hardware encryption device.
- RC=60      Error opening work/reject file.
- An error occurred when the specified file was being opened.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=61      Error reading work/reject file.
- An error occurred when the specified file was being read.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=62      Error writing work/reject file.
- An error occurred when the specified file was being written.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=63      Error closing work/reject file.
- An error occurred when the specified file was being closed.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=65      Unable to create work file.
- DECOMP** cannot generate a unique name for the work file.
- ACTION: Delete the temporary files (names beginning with '~WK') and rerun **DECOMP**.

- RC=66      Error deleting work file.
- An error occurred when the specified file was being deleted.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=70      No input files found.
- No input files were found to compress/decompress.
- ACTION: This may be a normal condition. However, the command-line parameters may specify an incorrect input path/file name. If this is the case, then correct the command-line parameters and rerun **COMPRESS** or **DECOMP**.
- RC=71      xxxx is an invalid parameter.
- The listed command line argument is not recognized as a valid parameter by **COMPRESS** or **DECOMP**.
- ACTION: Refer to the command-line parameters for this option description's correct syntax. Correct the command line and rerun **COMPRESS** or **DECOMP**.
- RC=72      String error processing command line arguments.
- An internal error occurred while parsing the command line.
- ACTION: Contact bTrade.com at 800-425-0444 for help in resolving this problem.
- RC=73      Invalid output path.
- The format of the output path specified in the command line is invalid.
- ACTION: Correct the command-line parameters and rerun **COMPRESS** or **DECOMP**.

RC=74            Cannot decompress files into themselves.

The command-line parameters specify that the output files are to retain the names of the input files, but the target directory for the output files is the same directory where the input files reside.

ACTION: Either specify a target directory that is different than the directory that contains the input files, or supply a file name for the output files. Refer to "Compressing on the PC (DOS, OS/2 and Windows)" on page 8-65 and "Decompressing on the PC (DOS, OS/2 and Windows)" on page 8-66 for details.

RC=75            Error generating automatic extension.

**COMPRESS** or **DECOMP** cannot generate an automatic extension for the output file. Files already exist for all possible extensions.

ACTION: Delete or rename some of the output files and rerun **COMPRESS** or **DECOMP**.

RC=76            Invalid file name in **SELECT**.

The **SELECT** parameter, which cannot be parsed, contains an invalid file name.

ACTION: Re-enter the command with a corrected **SELECT** parameter.

RC=77            Invalid file sequence in **SELECT**.

The **SELECT** parameter contains an invalid file sequence. The **SELECT** cannot be parsed.

ACTION: Re-enter the command with a corrected **SELECT** parameter.

RC=78            Error creating output directory.

An error occurred when the specified output directory was being created.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **DECOMP**.

RC=79            Error processing **TRANTBL**.

An error occurred when opening or reading the specified translate table file.

ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.

- RC=80      Error opening parameter file.
- An error occurred when opening the parameter file.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS**.
- RC=81      Error reading parameter file.
- An error occurred when reading the parameter file.
- ACTION: Examine the accompanying system message to determine the cause of the error. Correct the problem and rerun **COMPRESS** or **DECOMP**.
- RC=83      Error processing **SYSDIN** parms.
- An error occurred when processing the **SYSDIN** dataset.
- ACTION: Examine the secondary error message to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=84      Dynamic allocation error.
- An error occurred during dynamic allocation of the output dataset.
- ACTION: Use the error codes in the message and reference the MVS documentation to determine the cause of the error. Correct the problem and rerun **DECOMP**.
- RC=89      Invalid Comm-Press version.
- RC=90      Invalid Comm-Press version.
- The data was compressed with a version of Comm-Press that is either no longer supported, or that is later than that used to decompress.
- ACTION: If the Comm-Press version is no longer supported, then the sender must upgrade to the current version. Otherwise, you must upgrade to the latest version. Contact bTrade.com at 800-425-0444 to acquire the latest version.
- RC=91-98      CRC failure (reason code=91-98) for input file.
- DECOMP** detected a failure during cyclic redundancy checking. Processing continues with the next compressed file.
- ACTION: The compressed data has been altered, or corrupted, and cannot be accurately decompressed. **DECOMP** bypasses the corrupted file and continues decompressing with the next compressed file.

- RC=100      Too many subdirectory levels (maximum=100).
- You are trying to recursively process nested subdirectories but the maximum of 100 nested levels has been reached.
- ACTION: The operation cannot be performed. You must rearrange your subdirectories before compression so that no more than 100 levels exist.
- RC=101      Error initializing certificate file.
- An error occurred when **COMPRESS** tried to initialize the certificate run-time file.
- ACTION: Examine the accompanying operating system messages, as well as the secondary error message, to determine the reason for the init error. Correct the error and rerun **COMPRESS**.
- RC=102      Error initializing private key file.
- An error occurred when **COMPRESS** tried to initialize the private key run-time file.
- ACTION: Examine the accompanying operating system messages, as well as the secondary error message, to determine the reason for the init error. Correct the error and rerun **COMPRESS**.
- RC=103      Error processing **SECFILE**.
- An I/O or other error occurred when **COMPRESS** was processing **SECFILE**.
- ACTION: Examine the accompanying operating system messages, as well as the secondary error message issued by **COMPRESS**, to determine the reason for the error. Correct the error and rerun **COMPRESS**.
- RC=104      Error processing **CPLOOKUP.TBL**.
- An I/O or other error occurred when **COMPRESS** was processing **CPLOOKUP**.
- ACTION: Examine the accompanying operating system messages, as well as the secondary error message issued by **COMPRESS**, to determine the reason for the error. Correct the error and rerun **COMPRESS**.

RC=105      Error processing random object.

A severe error occurred when **COMPRESS** or **DECOMP** was processing the random number object.

ACTION: Examine the secondary error message issued by **COMPRESS** to determine the reason for the error. The BSAFE return code is especially important in determining the cause of the error. "BSAFE Return Codes" appear on page 10-112. Contact bTrade.com for help in resolving the error.

RC=106      Error building SxS segment.

A severe error occurred when **COMPRESS** was building the SxS segment used to hold bulk encryption information.

ACTION: Examine the accompanying operating system messages, as well as the secondary error message issued by **COMPRESS** to determine the reason for the error. The BSAFE return code is especially important in determining the cause of the error. "BSAFE Return Codes" appear on page 10-112. The most common reason for this error is the absence of a valid certificate for the **RECEIVER**. Correct the error and rerun **COMPRESS**.

RC=107      Error building signature segments (SxA/SVA) .

A severe error occurred when **COMPRESS** was building the SxA/SVA segments used to hold digital signature information.

ACTION: Examine the accompanying operating system messages, as well as the secondary error message issued by **COMPRESS** to determine the reason for the error. The BSAFE return code is especially important in determining the cause of the error. "BSAFE Return Codes" on page 10-112. The most common reason for this error is the absence of a valid certificate or private key for the **SENDER**, or a missing or invalid passphrase file. Correct the error and rerun **COMPRESS**.

RC=108      Error processing SxS segment.

A severe error occurred when **DECOMP** was processing the SxS segment used to hold bulk encryption information.

ACTION: Examine the accompanying operating system messages, as well as the secondary error message issued by **DECOMP**, to determine the reason for the error. The BSAFE return code is especially important in determining the cause of the error. "BSAFE Return Codes" appear on page 10-112. The most common reason for this error is the absence of a valid certificate or private key for the **RECEIVER**, or a missing or invalid passphrase file. Correct the error and rerun **DECOMP**.

RC=109      Error processing signature segment (SxA/SVA) .

A severe error occurred when **DECOMP** was processing the SxA/SVA segments used to hold digital signature information.

ACTION: Examine the accompanying operating system messages, as well as the secondary error message issued by **DECOMP**, to determine the reason for the error. The BSAFE return code is especially important in determining the cause of the error. "BSAFE Return Codes" appear on page 10-112. The most common reason for this error is the absence of a valid certificate for the **SENDER**, or the data has been corrupted or tampered with. Correct the error and rerun **DECOMP**.

RC=110      Error initializing symmetric key file.

An error occurred when **COMPRESS** tried to initialize the symmetric key run-time file

ACTION: Examine the accompanying operating system messages, as well as the secondary error message, to determine the reason for the init error. Correct the error and rerun **COMPRESS**.

RC=111      Error verifying message authentication code (SxE).

The computed message authentication code did not match what was sent on the SxE segment.

ACTION: The decompressed data has failed authentication processing. This may be due to a corrupted file or tampering. **DECOMP** bypasses the file and continues decompressing with the next compressed file.

RC=112      Unsecured group (GS/GE) in input.

The EDI data contains an unencrypted group.

ACTION: The **SECURECK** option was specified to prevent unencrypted EDI data from being written to **DATAOT**. The EDI envelope containing the unencrypted group has been written to the reject file **DATAXX**.

RC=113      Unsecured transaction (ST/SE) in input.

The EDI data contains an unencrypted transaction.

ACTION: The **SECURECK** option was specified to prevent unencrypted EDI data from being written to **DATAOT**. The EDI envelope containing the unencrypted transaction has been written to the reject file **DATAXX**.

- RC=114      Error initializing participant file.
- An error occurred when **COMPRESS** tried to initialize the participant table run-time file
- ACTION: Examine the accompanying operating system messages, as well as the secondary error message, to determine the reason for the init error. Correct the error and rerun **COMPRESS**.
- RC=115      No relationship found in lookup table.
- The **SECUREONLY** option was specified, but a security relationship record was not found in the lookup run-time file.
- ACTION: Examine the secondary error message to determine the participants for which no relationship exists. A security relationship must be defined in TDManager and new runtime files must then be installed, to secure the data.
- RC=116      Caused by permissions, DB2 not starting or invalid subsystem name.
- ACTION: Look up the specified DB2 return code for the correct action.
- RC=117      Mixing EDIFACT message types not allowed.
- ACTION: Refer to EDIFACT Standards.
- RC=118      Error building AUTACK message. Caused by an environmental error, inadequate permissions, insufficient storage, public key error, or run-time files are not current.
- ACTION: A secondary message will help identify the source of the problem.
- RC=119      Error building AUTACK message. Caused by an environmental error, inadequate permissions, insufficient storage, public key error, or run-time files are not current.
- ACTION: A secondary message will help identify the source of the problem.

## Secondary Messages/Return Codes

The following secondary messages and return codes may appear along with one of the previous messages.

RC=1	Error allocating memory
RC=2	Record not found
RC=3	Initialization error
RC=4	Seed error
RC=5	Message digest does not match SVA
RC=6	Error encrypting digest
RC=7	Signature does not verify
RC=8	Error making random initialization vector
RC=9	Error making random encryption key
RC=10	Error encrypting random encryption key
RC=11	Error decrypting random encryption key
RC=12	Error getting public key
RC=13	Error getting private key
RC=14	No valid certificate found
RC=15	Unsupported compression algorithm
RC=16	Unsupported encryption algorithm
RC=17	Unsupported assurance algorithm
RC=18	Unsupported filter algorithm
RC=19	Error getting secret authentication key
RC=20	Error getting secret encryption key
RC=21	Error retrieving onetime key
RC=22	Encryption hardware error

RC=23	Missing AUTACK message
RC=24	Invalid AUTACK message
RC=60	Error opening file
RC=61	Error reading file
RC=62	Error writing file
RC=70	Incomplete segment

## AUTACK Error Codes

The following error codes can be returned when an error is encountered processing an AUTACK message in an EDIFACT interchange:

RC=1000	Error allocating memory
RC=1001	Invalid AUTACK version number
RC=1002	Invalid AUTACK release number
RC=1004	Invalid AUTACK format
RC=1005	NVB digest does not verify
RC=1006	Invalid NVB signature block
RC=1007	Invalid NVB contract number
RC=1008	Invalid NVB EB authorization number
RC=1009	Invalid NVB signature date
RC=1010	Invalid NVB signature time

## AUTACK Version 1.1 Error Codes

The following error codes are specific to a version 1.1 AUTACK message:

RC=1100	Invalid segment
RC=1101	Invalid version number
RC=1102	Invalid release number
RC=1103	Invalid security reference value
RC=1104	Invalid reference value
RC=1105	Invalid signature 1
RC=1106	Invalid signature 2
RC=1107	Invalid interchange value
RC=1108	Missing UNH segment
RC=1109	Missing USH segment
RC=1110	Missing USR segment
RC=1113	Invalid UNH segment
RC=1114	Invalid USH segment
RC=1115	Invalid USR segment

## AUTACK Version 4.1 Error Codes

The following error codes are specific to a version 4.1 AUTACK message:

RC=4100	Invalid segment
RC=4101	Invalid version number
RC=4102	Invalid release number
RC=4103	Invalid security reference value
RC=4104	Invalid reference value

RC=4105	Invalid signature 1
RC=4106	Invalid signature 2
RC=4107	Invalid interchange value
RC=4108	Missing UNH segment
RC=4109	Missing USH segment
RC=4111	Missing USC segment
RC=4112	Missing USY segment
RC=4113	Invalid UNH segment
RC=4114	Invalid USH segment
RC=4116	Invalid USC segment
RC=4117	Invalid USY segment

## Additional API Error Codes

The following additional error codes can be returned in the RC field by the API:

Error Code	Description
13	Input buffer too short (DCMPPROG)
11	Output buffer too short (COMPPROG)
16	Invalid compression signature (DCMPPROG)

## BSAFE Return Codes

BSAFE return codes are issued by the RSA public/private key functions. Most of the return codes experienced should be explained in the message that accompanies them. However, if they aren't, then they are described below. If you receive a BSAFE return code that is not explained by the error message and does not appear in the following list, contact bTrade.com for help in resolving the problem.

Return Code	Description
256	Insufficient memory
257	Invalid signature on certificate or CRL
258	Invalid attributes object
259	Invalid number of values for the attribute type
260	Requested attribute type is not in the attributes object
261	Invalid attribute value tag
262	Unknown attribute type
263	Invalid attribute value
264	Invalid attribute value length
265	Invalid format for BER coding
266	Operation was canceled by the surrender function

267	Certificate chain could not be constructed
268	Invalid certificate encoding
269	Invalid certificate object
270	Invalid co set
271	Invalid CRL coding
272	Invalid CRL object
273	Generic data error
274	Fatal database interface error
275	Unsupported DEK (data encryption) algorithm
276	Unknown DEK (data encryption) algorithm
277	Invalid digest object
278	Fatal I/O interface error in enhanced text stream
279	End of stream
280	Even exponent not permitted in public or private key
281	Invalid exponent length in public or private key
282	Cryptographic hardware error
283	Syntax error in PEM header fields
284	Index out of range
285	Invalid length for input data
286	Fatal I/O interface error in input stream
287	Fatal I/O interface error
288	Invalid list object
289	Invalid internal memory object
290	Invalid signature on message
291	Invalid me set

292	Unsupported MIC (message digest) algorithm
293	Unknown MIC (message digest) algorithm
294	Invalid modulus length in public or private key
295	Invalid name object
296	Random object not seeded
297	Certificate, private key, or CRL not found
298	Recipient of incoming message not among potential recipients
299	Unsupported operation requested
300	Invalid length for output data
301	Fatal I/O interface error in output stream
302	Data block exceeds 32,767 bytes
303	Invalid parameter
304	Invalid password for decrypting data
305	Unsupported password-based encryption algorithm
306	Unknown password-based encryption algorithm
307	Fatal I/O interface error in PKCS input stream
308	Fatal I/O interface error in PKCS output stream
309	Fatal I/O interface error in PKCS stream
310	Invalid private key format
311	Invalid message process type
312	Invalid encoding of protected data
313	Invalid public key format
314	Invalid random object
315	Unsupported certificate or CRL signature algorithm

316	Unknown certificate or CRL signature algorithm
317	Invalid syntax for base 64 encoding
318	Fatal I/O interface error in text stream
319	Argument expected to be a #define'd constant invalid
320	Invalid certificate validity
321	Invalid message version
322	Invalid you set
512	Value of the algorithm object has already been set by a call to B_SetAlgorithmInfo or by an algorithm parameter generation
513	Invalid format for the algorithm information in the algorithm object
514	Algorithm object has not been initialized by a call to the Init procedure
515	Algorithm object has not been set by a call to B_SetAlgorithmInfo
516	Invalid algorithm object
517	Unknown operation for an algorithm or algorithm information type
518	Insufficient memory
519	Operation was canceled by the surrender function
520	Generic data error
521	Invalid even value for public exponent in keypair generation
522	Invalid exponent length for public exponent in keypair generation
523	Cryptographic hardware error
524	Invalid encoding format for input data
525	Invalid total length for input data
526	Value of the key object has already been set by a call to B_SetKeyInfo or by a key generation
527	Invalid format for the key information in the key object

528	Invalid key length
529	Key object has not been set by a call to B_SetKeyInfo or by a key generation
530	Invalid format for the key information in the key object
531	Unknown operation for a key info type
532	Invalid internal memory object
533	Unsupported modulus length for a key or for algorithm parameters
534	Algorithm is improperly initialized
535	Algorithm chooser does not support the type of key information in the key object for the specified algorithm
536	Maximum size of the output buffer is too small to receive the output
537	Data block exceeds 32,767 bytes
538	Random algorithm has not been initialized by a call to B_RandomInit
539	Invalid algorithm object for the random algorithm
540	Signature does not verify
541	Required algorithm information is not in the algorithm object
542	Required key information is not in the key object
543	Update called an invalid number of times for inputting data
544	Algorithm chooser doesn't contain the algorithm method for the algorithm specified by the previous call to B_SetAlgorithmInfo update called an invalid number of times for outputting data

# Appendix D

## Support CCA

### Overview

This optional feature of TDCompress Enhanced provides support for Common Cryptographic Architecture (CCA) hardware devices (examples are the IBM 4753 and the cryptographic co-processor).

This technology allows off-loading of processor cycles to the cryptographic unit. The implementation retains the ability to use TDCompress software when the hardware is not available.

Key values are secured in the TDManager tables and are passed to the CCA interface at execution time.

### Prerequisites

CCA support is available in the MVS(OS390) environment.

Special requirements in addition to those specified for the Comm-Press basic package:

- CCA hardware device
- Started task driver for CCA device
- ICSF for OS/390 Version 2.8 and above

### Operation

Specify use of hardware encryption devices by adding the 'CCA' option to the PARM clause on the EXEC JCL statement. The 'CCA' option is valid for both the COMPRESS and DECOMP programs. For Example:

```
//STEP010 EXEC PGM=COMPRESS,PARM=' SECURE,CCA'
```

# Glossary of Terms

algorithm	A clearly specified mathematical process for computation; a set of rules which gives a prescribed result.
alphanumeric	A character set that contains both letters and digits.
ANSI	American National Standards Institute; ANSI is a private, non-profit organization responsible for the development and approval of voluntary consensus standards in the United States. ANSI approves standards developed primarily by trade, technical, professional, consumer, and labor organizations. They approve standards only when verified evidence is presented by a standards developer that those affected by the standard have reached substantial agreement on its provisions.
ASC X12	ANSI Accredited Standards Committee (ASC) X12 chartered to develop uniform standards for electronic interchange of business transactions.
ASC X12.58	The ASC X12 subcommittee that defines standards for securing X12-formatted EDI data. The standards address authentication, encryption, and verification of the security originator to the security recipient.
ASCII	American Standard Code for Information Interchange; A code for representing characters as numbers, with each character assigned a number from 0 to 127.
asymmetric algorithm	An encryption algorithm that uses two mathematically related, but different, key values to encrypt and decrypt data. One value is designated as the private key and is kept secret by the owner. The other value is designated as the public key and is shared with the owner's trading partners. The two keys are related such that when one key is used to encrypt, the other key must be used to decrypt.
authentication	The verification of the source, uniqueness, and integrity of a message.
certificate request	An uncertified public key created by a trading partner as part of the RSA keypair generation. The certificate request must be approved by a CA, (be issued as a certificate) before it can be used to secure data.

certificate	A certified public key. Certificates are issued by a CA, which includes adding the CA's distinguished name, a serial number and starting and ending validity dates to the original request. The CA then adds its digital signature to complete the certificate.
Certifying Authority	Entity responsible for issuing certificates. Before issuing a certificate, the CA follows published policies to verify the identity of the trading partner that submitted the certificate request. Once issued, other trading partners can trust the certificate based upon the trust placed in the CA and its published verification policy. TDManager is the CA product used with TDCompress.
ciphertext	Encrypted data.
CRLF	carriage return/line feed
delimiter	A field separator (for example, comma, tab, or other defined character)
decryption	The process of transforming ciphertext into plaintext.
DES	Digital Encryption Standard; A U.S.government standard encryption algorithm that has been endorsed by the U.S. military for encrypting "unclassified but sensitive" information. It is a symmetric algorithm; the same key is used for encryption and decryption.
digital signature	Electronic signature that can be applied to any electronic document. An asymmetric encryption algorithm, such as the RSA algorithm, is required to produce a digital signature. The signature involves hashing the document and then encrypting the result with the sender's private key. Any trading partner can verify the signature by decrypting it with the sender's public key, recomputing the hash of the document, and then comparing the two hash values for equality.
DISA	Data Interchange Standards Association: Secretariat for ASC X.12.
EBCDIC	Extended Binary-Coded Decimal Interchange Code; An IBM code for representing characters as numbers. Although widely used on large IBM computers, most other computers, including PCs and UNIX workstations, use ASCII codes.
EC	Electronic Commerce. The use of information technologies to conduct business between trading partners. Electronic Commerce technologies include those which exchange data (EDI, email), access data (shared databases, bulletin boards), and automatically capture data (bar coding).

EDI	Electronic Data Interchange: The inter-organizational, computer-to-computer exchange of business documentation in a standard, machine-processable format.
EDI data element	The smallest meaningful piece of information in an EDI transaction. The data element equivalent to a field in a database or paper document that condenses lengthy descriptive information into short code. Data segments are made up of data elements.
EDI data segment	A data segment is made up of data elements, which occur in a specific sequence as defined by an EDI standard (X.12). A data segment is the equivalent to a record in a database or paper document.
EDIFACT	United Nations Electronic Data Interchange for Administration, Commerce, and Transport; International standard set by the UN and administered in the U.S. by DISA. This standard has been widely implemented in western Europe.
EDI name	A unique identifier used by communications software and public networks for addressing and routing files.
encryption	The process of transforming plaintext into ciphertext.
hub	A large company with a highly-developed EDI program that actively encourages EDI implementation and development among its vendors and other business partners.
key	Cryptographic key. A parameter that determines the transformation from plaintext to ciphertext or vice versa. For example, a <b>DES</b> key is a 64-bit parameter consisting of 56 key bits and 8 bits, which may be used for odd parity.
key interval	The time period for which a key will be active.
MAC	Message Authentication Code. A cryptographically computed value that is the result of passing text or numeric data through the authentication algorithm using a specific key.
passphrase	A string of 64 characters used to encrypt private keys. Passphrases are randomly generated during the key generation process. They may be stored with the private key or written to a separate file when the TDManager run-time files are imported.
plaintext	Unencrypted data; intelligible data that can be directly acted upon without decryption.

private key	The mathematical value of an asymmetric key pair that is not shared with trading partners. The private key works in conjunction with the public key to encrypt and decrypt data. For example, when the private key is used to encrypt data, only the public key can successfully decrypt that data.
public key	The mathematical value of an asymmetric key pair that is shared with trading partners. The public key works in conjunction with the private key to encrypt and decrypt data. For example, when the public key is used to encrypt data, only the private key can successfully decrypt that data.
RC2	A variable key size block cipher, designed to be a replacement for <b>DES</b> .
receiver	The receiving trading partner, system or process that is the destination of transmitted data.
secret key	The value used in a symmetric encryption algorithm to both encrypt and decrypt data. Secret keys must be known only by the trading partners authorized to access the encrypted data.
sender	The sending trading partner, system or process that is the originator of transmitted data.
session key	A random, one-time secret key.
symmetric algorithm	An encryption algorithm that uses the same secret key to both encrypt and decrypt data.
trading partner	A supplier, customer, service provider, or other party with whom business documents are routinely exchanged.
VAN	Value Added Network; The source or service that resolves the issues resulting from communicating with a number of different trading partners. They provide EDI communication skills, expertise, and equipment necessary to communicate electronically.

# Index

## A

ACF2.....4-34  
 API.....6-55, 10-89  
 ASC X12.58 standards  
   supported by Comm-Press2000 .....2-12  
 ASCII .....6-46  
   translation to EBCDIC.....1-8  
 Authentication  
   using digital signatures .....2-17  
**AUTOEXT** .....**3-24**

## B

batch key generation .....4-32  
 bind the plan.....6-43  
 BSAFE return codes .....10-112

## C

certificate file .....4-33  
 Comm-Press2000  
   benefits .....1-5  
   upgrading to new versions .....1-9  
 compprog .....10-89  
 Configuration  
   run-time options.....1-8  
 Conversion ASCII/EBCDIC .....10-93  
 CRLF  
   run-time option.....1-8

## D

DATAIN .....10-84  
 DATAIN DD.....6-45, 6-46  
 DATAOT .....10-84, 10-85  
 DATAOT DD .....6-45, 6-46  
 DE3 .....2-18  
 DELIMIT .....2-13, **3-25**  
 Delimiters .....1-9  
 DES .....2-18, 4-34  
 digital signature.....2-17, 4-34  
 digital signatures .....2-14  
 DIRNAME.....8-65, 9-76  
 Distinguished Name .....4-33  
 DOS.....8-63, 8-64  
 DOS/Windows .....8-63  
 dynamic allocation .....6-48

## E

easyacc.ini .....4-32  
 EBCDIC  
   translation to ASCII .....1-8  
 EDI.....**3-26**, 8-73

  security options .....2-12  
   standard formats supported.....1-6  
 EDIFACT.....8-73  
 Encryption algorithms .....1-6  
 error messages .....10-95  
 Example Non-EDI.....5-41, 8-72  
 Example X12-EDI.....8-70  
 Execution options  
   KEEPSIGS .....2-12  
   listing .....2-12  
 Extended Security Option.....1-5  
   API .....10-89

## F

Filtering .....2-17

## G

GENKEYS .....4-32, 8-68  
 GENKEYS JCL .....6-53, 10-86  
 GS segment .....6-53, 8-70

## H

hash .....2-17

## I

IMPORT.....10-83  
 Installation  
   upgrading to new versions .....1-9  
**IV 3-26**  
**IV=** .....**2-18**

## J

JCL.....6-43  
 Job Control Language .....1-9

## K

KEEPSIGS .....2-13  
**KEY** .....**3-26**  
 Key generation  
   methods .....4-32  
**KEY=** .....**2-18**

## L

Licenses  
   software .....1-6  
 load modules .....6-44  
 Log files  
   created by Comm-Press2000 .....2-12  
 LOGPATH.....2-13, **3-27**

## M

MD5	
hashing algorithm.....	2-17
message digest.....	2-17
MVS Installation	
JCL to unload tape.....	6-43
using a tape cartridge .....	6-43
using diskette or CD-ROM .....	6-42
MVS-DB2.....	6-43

## N

NOINFO .....	8-65, 9-76
non-EDI data.....	6-54, 8-72

## O

Operating Systems	
supported by Comm-Press2000 .....	1-7
transferring files between.....	1-9
OS/2 .....	8-64
OS/400 .....	10-82, 10-83

## P

PARM field.....	6-45
passphrase.....	4-34, 6-55, 8-72
PC file .....	8-68
PC/Windows .....	8-65
PDS .....	6-42, 6-46, 6-56
<b>PF</b> .....	<b>3-28</b>
Public/private keys	
key technology.....	2-14
technology overview.....	2-15

## R

RACF .....	4-34
RC2 .....	2-18, 4-34
RECEIVER .....	2-13, <b>3-28</b>
RECURSE.....	8-65
RSTLIB .....	10-82
Runtime files .....	4-32
Run-time files	
private key .....	4-33
Run-time files	
symmetric key .....	4-34
Run-time files	
lookup table .....	4-34
Run-time files	
participant table .....	4-35
Run-time options	
CRLF .....	1-8, 1-9

RUNTIMEPATH.....	2-13, <b>3-28</b> , 4-34
------------------	--------------------------

## S

SAVLIB .....	10-82
SECFILE .....	2-13, <b>3-29</b> , 5-39
SECFILE keywords	
MVS example user-defined headers .....	5-37
PC example user-defined headers.....	5-39
secret key.....	2-19
Secret keys.....	7-60
algorithms .....	2-15
SECURE .....	2-13
TDManager .....	4-33
SECUREONLY .....	2-13
Security options	
EDI.....	2-12
<b>SELECT</b> .....	<b>3-29</b>
<b>SENDER</b> .....	<b>2-14, 3-29</b>
signature .....	6-45
Software	
licenses .....	1-6
<b>SQL</b> .....	<b>2-14, 3-30</b>
SQL parameter .....	6-43
Symmetric keys.....	2-15

## T

<b>TRANSID</b> .....	<b>2-13, 2-14, 3-30</b> , 5-40
translation table.....	6-56, 10-88
TSO RECEIVE .....	6-42
TSO TRANSMIT .....	6-42

## U

UCS EDI data.....	8-73
UNIX.....	9-76
UNIX/AIX.....	9-77
UNTDI.....	8-73
UNWRAP.....	2-14
USEGS .....	2-14

## V

VMS.....	7-58, 7-59, 7-60
----------	------------------

## W

Windows .....	8-64
---------------	------

## X

X12.....	8-73
X12 data.....	6-53, 8-70